ON THE PRUNE AND SEARCH PARADIGM IN COMPUTATIONAL GEOMETRY

by

SHREESH JADHAV

DEPARTMENT OF COMPUTER SCIENCE

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

April 1994

Title of the Thesis

ON THE PRUNE AND SEARCH PARADIGM IN COMPUTATIONAL GEOMETRY

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

by
Shreesh Jadhav

to the

DEPARTMENT OF COMPUTER SCIENCE

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

April, 1994

CS-1994-D-JAD-PRU

CERTIFICATE

Certified that the work contained in the thesis entitled "*On the Prune and Search Paradigm in Computational Geometry*", by "*Shreesh Jadhav*", has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

[illegible] 11th April, 9

(Dr. Ashish Mukhopadhyay)
Department of Computer Science,
Indian Institute of Technology,
Kanpur.

April, 1994

Abstract

The present thesis provides algorithms for some problems in computational geometry using prune and search technique. We solve the intersection radius problems in the plane for sets of line segments, rays, wedges, half planes and convex polygonal disks. Then we compute a centerpoint of a planar set of points. This is followed by description of a generalised technique of prune and search.

The *intersection radius* of a set of n geometrical objects is the radius of the smallest closed ball that intersects all the objects of the set. We have designed algorithms to optimally solve intersection radius problems for various kinds of objects. We show how the prune and search technique, coupled with the strategy of replacing a ray by a point or a line can be used to solve, in linear time, the intersection radius problem for a finite set of line segments in the plane.

Next, the scope of this technique is enlarged by showing that it can also be used to find the intersection radius of a set of convex polygons in linear time. Moreover, it is immaterial if the set also contains other types of geometric objects like points, lines, rays, line segments, half planes and wedges. In fact, it is shown how such a mixed set of objects can be handled in a unified way; and this is the other important contribution of the thesis. Previously there existed no known algorithms to solve these intersection radius problems efficiently.

The *center* of a set $\mathcal{P}$ of n points in plane is defined as the set of points c such that any closed half plane containing c contains at least $\lceil n/3 \rceil$ points of $\mathcal{P}$. A *centerpoint* is any point in the center. It can be viewed as a generalisation of the median of a set of reals. In the thesis, it is shown how a centerpoint of a finite set of points in the plane is computed in linear time. The described algorithm is optimal which significantly improves the $O(n \log^3 n)$ complexity of the previously best known algorithm. We need to use suitable modifications of the ham-sandwich cut algorithm and the prune and search technique to achieve this improvement.

The optimal sorting network by Ajtai et al. has a noteable feature that it approximately sorts the input data in its intermediate steps. A technique is presented for solving computational geometry problems by exploiting this fact. This technique is a sequel to the Megiddo's technique of designing serial algorithms by applying parallel computation algorithms. We get optimal linear time algorithms for some problems by applying this technique. We do this by the synthesis of prune and search and parametric searching.

"Once there lived a village of creatures along the bottom of a great crystal river. Each creature in its own manner clung tightly to the twigs and rocks of the river bottom, for clinging was their way of life, and resisting the current what each had learned from birth. But one creature said at last, "I trust that the current knows where it is going. I shall let go, and let it take me where it will. Clinging, I shall die of boredom."

The other creatures laughed and said, "Fool! Let go, and that current you worship will throw you tumbled and smashed across the rocks, and you will die quicker than boredom!"

But the one heeded them not, and taking a breath did let go, and at once was tumbled and smashed by the current across the rocks. Yet, in time, as the creature refused to cling again, the current lifted him free from the bottom, and he was bruised and hurt no more.

And the creatures downstream, to whom he was a stranger, cried, "See a miracle! A creature like ourselves, yet he flies! See the Messiah, come to save us all!" And the one carried in the current said, "I am no more Messiah than you. The river delight to lift us free, if only we dare let go. Our true work is this voyage, this adventure."

But they cried the more, "Saviour!" all the while clinging to the rocks, making legends of a Saviour."

Preface

Geometry is one of the most ancient branches of mathematics. One of the earliest attempt to systematise Geometry was made as early as in 300 B.C. by Euclid in Greece. This field also has distinction of being the first ever subject to profess of being rational and logical. At least it was believed by the people at that time and even now of standing on strictly formal grounds of logic. The whole system in *Elements* by Euclid is built on five basic geometrical postulates and numerous theorems are proved using only these few postulates. It can be said without any doubt that Geometry is one of the oldest field and scores of brilliant mathematicians and researches have worked in this field enriching geometry considerably by their discoveries and findings.

Computational Geometry on the other hand is only of recent origin. It deals with the computational aspects, its theory and applications. The subject matter of Computational Geometry ranges from theoretical computer science to that of more applied nature in Algorithms. To categorise the field might be difficult but it can be safely presumed that Computational Geometry is an offshoot of Applied Mathematics. It profusely uses the results of Combinatorial Geometry and constructs of Euclidean Geometry in its theoretical framework and the results of Analytical Geometry in its applications. When it deals with proofs of existences, theorems concerning Geometry, classifications of geometrical objects in different equivalent classes or enumerations of these then we are concerned with theoretical Computational Geometry and when it deals with computing geometrical objects, or their attributes or their count then we are concerned with application of Computational Geometry.

It is important to know the types of problems in Computional Geometry to understand it better. The fundamental problems are that of construction of convex hulls, location of points in subdivisions, construction of Voronoi diagrams, etc. For a more complete list the pioneering doctoral work of Shamos can be referred.

In this thesis we are concerned with the algorithmic aspects of Computational Geometry. We give algorithms to compute geometrical concepts such as intersection radius and center points. Intersection radius has got reference in the first ever collection of problems in Computational Geometry by Shamos where as the concept of center is dealt with in the book on geometry by Yaglom and Boltyanskiĭ. The last part of the thesis deals with a generalised computational tool that can be used for geometrical optimisation problems.

I wish to acknowledge my thesis supervisor Prof. Ashish Mukhopadhyay for his guidance during my Ph. D. work. I would like to acknowledge the work of researchers (especially Herbert Edelsbrunner and Nimrod Megiddo) before me in the field of computational geometry for providing motivation to persue this work. I thank my parents for leaving me alone to persue higher studies. I thank all my friends in and out of I.I.T. Kanpur for the company. I thank the I.I.T. Kanpur administration for bearing me. I thank the workers' cooperative of I.I.T. Kanpur for allowing me a chance to do social work. I finally thank myself for completing this thesis.

Contents

List of Figures

Chapter 1

Introduction

This thesis deals with a method of computing intersection radii for various kinds of geometrical objects and also of computing a centrepoint for a planar set of points. It also describes a general technique for applying the prune and search paradigm for solving geometric optimisation problems.

1.1 Intersection Radius

Suppose there are several sites in a two-dimensional metric space and we have to choose a point p such that it is "nearest" to all the sites in question. Here the word "nearest" implies that the point p minimises the maximum distance of the sites from p. If a circle is drawn to "span" (cover) all the sites then it is called a *spanning* circle. When we draw a spanning circle centered on the "nearest" point p with radius equal to its distance from the furthest site(s), then this circle is called the *minimum spanning circle.*

Phrases such as "spanning circle" and "minimum spanning circle" are meaningful so long as we are interested in spanning only point sites. These phrases are inadequate to describe the problem of computing a smallest closed disk that intersects other objects such as lines, rays, line segments, etc. Thus the notion of stabbing is introduced.

When one object intersects or touches another it is said that the former *stabs* the latter. An object is called a *stabber* of a given set of objects if it stabs each object of this set. For example, the closure of the interior of the minimum spanning circle above is the smallest closed disk that stabs the given set of points.

Let C be a finite collection of geometrical objects in a d-dimensional Euclidean space. The *stabbing* problem consists of finding an object (the stabber) which intersects each member of C. Typically, the stabber could be a line, a hyperplane or a disk, and C could be a collection of points, lines, line segments, rays, hyperspheres, polyhedron or any mix of these. A survey of some recent results is available in the paper by Houle et al. [15].

A classical problem in this area is that of finding the intersection radius of a finite set of points in the plane, which is also known as the *1-centre* problem [30, 31, 6]. It was shown by Megiddo, and subsequently by Dyer [22, 12, 11], how this can be solved in linear time for fixed d. However, until a recent paper [3] no attempt was made to extend this to a more complex set of objects than points or to a collection containing different kinds of objects.

Attempts have been also made recently to find more complicated stabbers for the stabbing problem, or to find the "best" stabber which optimises some measure defined on the class of stabbers in question. Goodrich and Snoeyink [14] presented an $O(n \log n)$ algorithm to find a convex polygon whose boundary intersects each of n parallel line segments. Rappaport and Meijer [24] showed that a perimeter minimising polygonal disk that intersects each of n parallel line segments can be found in $O(n \log n)$ time. They have also extended their result to a set of isothetic line segments. Mukhopadhyay and Kumar [27] have shown that for a set of parallel line segments an area minimising polygonal disk can also be found in $O(n \log n)$ time. Bhattacharya and Toussaint[4] gave an $O(n \log^2 n)$ algorithm for computing the shortest line segment that intersects a set of n given line segments in the plane. This bound was later improved to $O(n \log n)$ by Bhattacharya et al.[2].

In the present thesis, the stabber is a disk. The stabbing problem in this case is known as the intersection radius problem[13]. The *intersection radius* of a planar collection of objects is the radius of the minimum stabbing disk of this collection. There is no known algorithm which solves the intersection radius problems for line segments. There are some variants of this problem which have been the subject of research of various researchers. As for example, when the stabber is a vertical line segment and the objects to be intersected are lines, the intersection radius problem is simply the Chebyshev approximation of points in the dual space[5, 29]. This problem can easily be solved in linear time by transforming it into a linear programming problem.

In this thesis, the intersection radius of a collection of line segments is computed by combining the prune-and-search strategy of Megiddo [21] with the strategy of replacing line segments with points or lines [3]. The scope of this technique is enlarged by showing that the intersection radius of a collection of convex polygons can also be computed in linear time. Further, it is immaterial if the collection contains a mix of other geometric objects such as lines, points, rays etc. for we show that it is possible to treat such a mixed collection of objects in a unified way.

1.2 Centrepoint of a Planar Set of Points.

We all have an intuitive idea as to what phrases like "the very center of the square" or "the very center of the city" mean. To capture this intuition in a quantitative way, the *center* of a set of n points, $\mathcal{P}$, in $\Re^d$ is defined as the maximal subset of $\Re^d$ such that any closed halfspace intersecting this subset contains at least $\lceil n/(d+1) \rceil$ points of $\mathcal{P}$ [35]. This subset is non-empty for any finite configuration of points (see, for example, [13]). Furthermore, it is closed and convex. A *centrepoint* is a member of the center of $\mathcal{P}$.

On the real line $\Re$, a centrepoint is no other than a median of $\mathcal{P}$. Thus a centrepoint can be viewed as a generalisation of the median of a set of reals. On the other hand, the center can also be viewed as a particular k-hull of $\mathcal{P}$. The *k-hull* of $\mathcal{P}$ is a maximal subset (closed and convex) of $\Re^d$ such that any closed halfspace intersecting this subset contains at least k points of $\mathcal{P}$. For instance, the 1-hull of $\mathcal{P}$ is its convex hull and the center is its $\lceil n/(d+1) \rceil$-hull. The property of balanced partitioning makes the centrepoint useful for efficient divide and conquer algorithms in geometrical computing and large scale scientific computing[26, 32, 25, 36]. Recently Donoho and Gasko have suggested that centrepoint can be used as "robust" and high "breakdown point" estimators for multivariate datasets [10].

The interesting algorithmic problem of computing a centrepoint has been considered by various researchers. Cole et al. gave an $O(n \log^5 n)$ algorithm for computing a centrepoint of a planar set of points [9]. Subsequently, Cole improved this bound to $O(n \log^3 n)$, using the powerful technique of slowing down a sorting network [8]. In this paper, we propose an optimal linear time algorithm for computing a centrepoint of a planar set of points by using suitable modifications of the ham-sandwich cut algorithm for a pair of separable point sets [23] and the prune and search technique of Megiddo [21].

Linear time algorithms, however, were known for computing an *approximate* or *ε-centrepoint* [19, 32, 23]. We obtain this weaker type of centrepoint if we decrease the lower bound, in the above definition of the center, to $\lceil n(1-\varepsilon)/(d+1) \rceil$, where $0 < \varepsilon < 1$. Actually, Megiddo [23] only gave an algorithm for computing a partition of a (planar) set of n points with two lines such that each closed quadrant contains at least $\lfloor n/4 \rfloor$ points. An algorithm for computing an ε-centrepoint, where $0 < \varepsilon \leq 1/4$, is implicit in this.

The thesis proposes an optimal algorithm for computing a centrepoint of a planar set of points by using an interesting modification of Megiddo's prune-and-search technique[21]. This consists of adding a few extra points in each pruning step so that a subspace of the original solution space is retained, while ensuring a net deletion of points. In the description of our algorithm, we assume the usual RAM model of computation; the point set $\mathcal{P}$, however, is not assumed to be in general position.

1.3 Prune and search in Slowed Down Sorting Networks

A substantial part of Computational Geometry deals with designing efficient algorithms for given problems. More often then not, the solutions of these depend on the ingenuity of the inventor of the algorithm. However, there are also attempts to provide general tools to solve the problems of a specific type. Paradigms like divide and conquer, parametric searching, line sweep, prune and search, linear programming etc. go a substantial step towards this goal.

The present thesis provides such a tool. We modify the method of parametric searching and slowed down sorting networks by Cole [8] to get a new technique.

The parametric searching of Megiddo [20] is as follows — Let there be an efficient parallel algorithm for a problem $\mathcal{A}$ such that solution of $\mathcal{A}$ can be used in the solution of another problem $\mathcal{B}$. Then, in some cases, we get an efficient sequential algorithm for $\mathcal{B}$ by exploiting the efficient parallel mechanism of the parallel algorithm for $\mathcal{A}$. This technique has been applied to a wide variety of problems yielding efficient algorithms. In particular, we achieve good results for the parametrised problems that use parallel sorting algorithms in their solutions. We do this by replacing the evaluation of the parallel comparisons of an iteration in the parallel version by simultaneous resolution of these in the serial version. The running time of these algorithms is further improved by the introduction of weights in the comparisons [8]. We then simultaneously evaluate at least a fraction of total weight of these in every iteration to design more efficient algorithms.

We further improve the running time of the above algorithms by introducing, wherever applicable, prune and search in these. We do this by seeking to compute the k-th largest element of the input set instead of seeking to sort it as in the previous techniques. For this, we run the sorting algorithm on a given input for a few iterations and then prune the set. It can be easily seen that this approach is useful only where pruning is applicable.

1.4 Organisation of Thesis

The organisation of thesis is as follows. In chapter 2 we solve the intersection radius problem for planar line segments in linear time. This algorithm is applied to all kinds of objects such as wedges, rays, halfplanes and convex polygonal disks in the next chapter. Moreover, it is shown how such a mixed set can be treated uniformly. In chapter 4, we present a linear time algorithm for computing a centrepoint of a finite planar set of points. We present a generalised tool for applying prune and search to a given problem in chapter 5. We conclude and discuss directions for further research in the last chapter.

Chapter 2

Intersection Radius of a Set of Line Segments

The *intersection radius* of a finite set of geometrical objects in a d-dimensional Euclidean space, E^d, is the radius of the smallest closed ball that intersects all the objects of the set. In this chapter, we describe optimal algorithms for some intersection radius problems in the plane. We first briefly review a linear time algorithm by Megiddo to determine the smallest disk that intersects a finite set of points. Next we show how the prune and search technique, coupled with the strategy of replacing a ray by a point or a line can be used to solve, in linear time, the intersection radius problem for a finite set of line segments in the plane.

Previously, no efficient algorithm was known to solve the intersection radius problem for line segments in the plane.

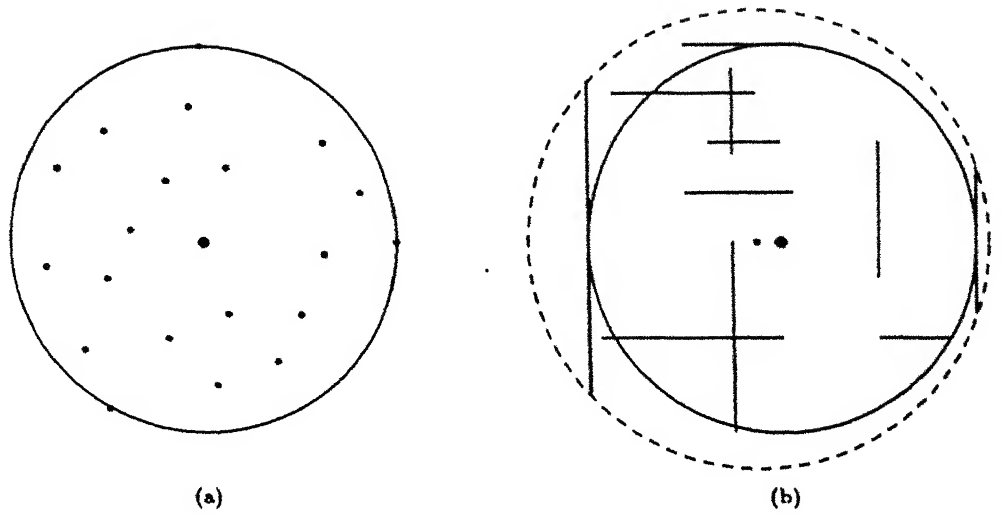

Figure 1: Spanning circles and stabbing disks of points and line segments

2.1 Introduction

Suppose there are several sites in a two-dimensional metric space and we have to choose a point p such that it is "nearest" to all the sites in question. Here the word "nearest" is to be understood in the following way: the nearest point p minimises the maximum distance from the sites to p. This problem frequently arises in practice in the installation of centralised facilities e.g. such as installation of a transmission centre. If a circle is drawn centred at p with radius equal to the distance from p to the farthest site, to span all the sites, then it is called a *minimum spanning circle* of the sites (Fig. 1(a)).

The phrase "spanning circle" is meaningful only if we are interested in spanning or covering points and is inadequate to describe the problem of computing the smallest closed disk that intersects other objects such as lines, rays, line segments, etc. Thus we introduce the notion of stabbing for these objects.

When one object intersects or touches another we say that the former *stabs* the later. A *stabber* of a set of objects is an object that stabs each member of this set. For example, the closure of the interior of the minimum spanning circle above is the *smallest stabbing disk* of the given set of points.

The concepts of spanning circles and stabbing disks, though related, are different in several ways. To span a set we only need to consider extreme points of the individual members of the set. For example, to span a set of line segments we only need to compute a spanning circle for the endpoints of these. However, this does not help in the computation of minimum stabbing disk of line segments which is frequently smaller. Similarly, in general also, it does not help to characterise the minimum stabbing disk if the minimum spanning circle of the given set is computed (Fig. 1(b)).

The *intersection radius* of a planar set of objects is the radius of the minimum stabbing disk of this set. Now this question naturally arises — how to compute the minimum stabbing disk/intersection radius for a given set of objects efficiently in linear time. This problem is progressively solved in this chapter and the next, first for simple objects, viz points and lines, and then for more complex ones.

The problem of computing the intersection radius usually reduces to a non-linear programming problem except when the stabbed set consists of straight lines alone. Not only for these but also for their counterparts in higher dimensions, i.e. hyperplanes, the intersection radius problem similarly reduces to a linear programming problem [3]. For fixed d, this latter problem can be solved in linear time, using the algorithm of Dyer or Megiddo or Clarkson [11, 21, 7]. Or instead, a recent simpler and more efficient randomised algorithm of Seidel [28] can be used that runs in $O(n \cdot (d+1)!)$ expected time where n is the number of constraints in the linear programming problem.

The case of points also has been fully tackled [21]. This was done by exploiting the fact that the bisectors of points are straight lines. These bisectors can be seen as linear functions and therefore the methods of linear programming problem had been applied to this case too.

The organisation of this chapter is as follows. In Section 2.2 we give a few definitions and discuss the motivation of the method to compute the intersection radius of points in linear time. The solution of the intersection radius problem for points in plane is presented in Section 2.3. In Section 2.4 we design algorithms to compute the intersection radius of lines and rays which are used to compute intersection radius of line segments in plane.

2.2 Preliminaries

Let $\mathcal{P}$ be a set of n points in the Euclidean plane and let c be the centre of a spanning circle of radius r. If $d(c,p)$ denotes the distance of c from a point p in $\mathcal{P}$, then clearly,

$$r \geq \max_{p \in \mathcal{P}} d(c,p)$$

Denoting by, $d(c,\mathcal{P})$, the distance of c from the set $\mathcal{P}$, which is also the radius of the smallest spanning circle centred at c, we have

$$d(c,\mathcal{P}) = \max_{p \in \mathcal{P}} d(c,p)$$

The *intersection radius*, $IR(\mathcal{P})$, of $\mathcal{P}$ is the radius of the minimum spanning circle. Thus,

$$IR(\mathcal{P}) = \min_{c \in J} d(c,\mathcal{P}).$$

We can similarly define the intersection radius of a set of lines, rays, line segments etc. We will denote minimum spanning circles by *min-circle* and minimum stabbing disks by *min-disk* . The centres of these in the unconstrained and constrained versions of the problem will be denoted by *unconstrained centre* and *constrained centre* respectively or just *centre* when the context is clear.

Let us first see how intersection radius of $\mathcal{P}$ is computed. Consider the min-circle of three points in the plane. The circumscribed circle of the triangle formed by

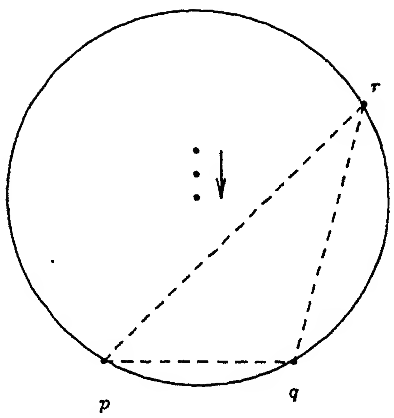

Figure 2: Computation of intersection radius of $\mathcal{P}$

these points is also a spanning circle of these. However, it is not necessary that this circle is a minimum too, unless the triangle formed by the points is an acute or right-angled triangle. In the case of obtuse-angled triangle the min-circle is obtained by drawing a circle with its diameter as the largest side of the triangle. A naive algorithm of complexity $O(n^3)$ to compute the intersection radius of $\mathcal{P}$ now is same as computing the radius of the largest of min-circles of all subsets of $\mathcal{P}$ of size three. It can be shown that the largest of these min-circles spans $\mathcal{P}$.

A better algorithm, that runs in $O(n^2)$ time, can be designed as an improvement to this naive algorithm by fixing two points in each iteration such that these two points are on the circumference of the spanning circles. These points are then dynamically updated until the min-circle of $\mathcal{P}$ is finally obtained. The details are as follows. We start with two fixed points, $p, q \in \mathcal{P}$, such that the points in $\mathcal{P}$ lie on the same side of $\overline{pq}$. The min-circles of three points are drawn for p, q and every other point of $\mathcal{P}$. Let the point corresponding to the largest of these circles, which incidently also spans $\mathcal{P}$, be r. If Δpqr is obtuse-angled then we delete the obtuse-angled vertex and repeat the same steps with the largest side as chord, otherwise the minimum spanning circle of Δpqr is the min-circle of $\mathcal{P}$(Fig 2).

We might think of improving this algorithm further by fixing only one point and then computing the spanning circles. But, since we need two more points to fix a spanning circle, and we have a quadratic number of choices, some other approach is needed. We therefore consider fixing a line on which the centres of the spanning circles are constrained to lie. Further, we also have to think of some way of not considering all of the quadratic choices that determine a spanning circle. This leads us to the important idea of prune and search.

The radius of the min-circle of $\mathcal{P}$ such that centres are constrained to be any point p is a convex function of p, $f(p)$. In particular, when the domain of this function is restricted to a line, $f(\cdot)$ is still convex. Therefore by examining the gradient of $f(\cdot)$ at any arbitrary point p on a query line we can tell whether $f(p)$ is minimum and if $f(p)$ is not minimum then on which side of p on the line the minimum of $f(\cdot)$ is attained. We will see in the next section how these observations help us in computing intersection radius of points in linear time.

2.3 Intersection Radius of Points in the Plane

When the objects to be intersected are only points, the algorithm of Dyer or Megiddo [12, 22] for the 1-centre problem can be used. A brief account of the algorithm is presented in this section, such that, it can be further generalised to include other kinds of objects.

Let $\mathcal{P}$ be the set of points whose intersection radius is to be determined and let its size be N_p.

To begin with the following question is answered: Which are the points that can be pruned. To answer this, we note that min-circle is determined by a set of only two or three points. These points are farthest from the centre of min-circle among all the points in $\mathcal{P}$. We try to determine these by removing the other points of $\mathcal{P}$ in an organised manner as follows:

If the centre of min-circle is contained in a half plane determined by the

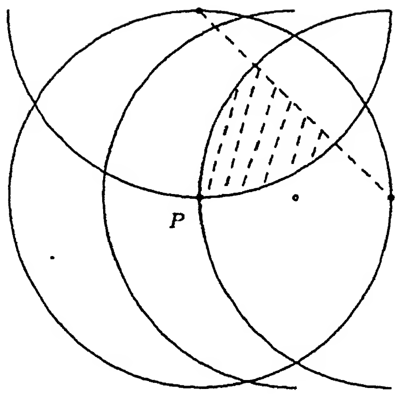

Figure 3: Localisation of Centre of Min-disk of $\mathcal{P}$

bisector of some two points of $\mathcal{P}$ then the point which is nearer to the centre can be dropped from $\mathcal{P}$ in the computation of intersection radius.

To prune away a constant fraction of points according to this rule, we need to localise the centre of min-circle with respect to a constant fraction of bisectors. The crux of the method is — How can this be done?

We further make note of the fact that, given a non optimum spanning circle of radius r, the radius of min-circle is smaller than r and therefore, the centre of min-circle will be at a distance smaller than r from any of the points in $\mathcal{P}$. This has the following consequences. First, for any point P, we can localise the centre in a small conical region whose apex is P. This can be done by first computing the smallest possible spanning circle centred at P and then computing intersection of interior of circles with radii equal to the radius of this circle centred at points on its circumference, of which at most three need to be considered for the purpose at hand (Fig. 3). Secondly, for any line L, we can localise the centre in one of the sides of L, by similarly computing the min-circle whose centre lies on L first and then computing the conical region, which can be shown not to intersect L, with respect to the centre of this circle.

To compute intersection radius of $\mathcal{P}$ in linear time we consider the following related subproblems. Subproblem 2 and Subproblem 3 can be solved using oracles of

Subproblem 1 and Subproblem 2 respectively. The convexity of the distance function is crucial in both the solutions as it also provides the direction, consequently a half space, in which the search for the centre of min-circle is localised.

Subproblem 1 *Compute the min-circle of $\mathcal{P}$ such that the centres of the spanning circles are at a fixed point P.*

Subproblem 2 *Compute the min-circle of $\mathcal{P}$ such that the centres of the spanning circles lie on a fixed line L.*

Subproblem 3 *Compute the min-circle of $\mathcal{P}$.*

The solution to the first subproblem is the simplest. We compute the maximum of the distances of points in $\mathcal{P}$ to the fixed point P and choose this distance as the radius of min-circle. This computation takes linear time.

The min-circle, whose centre is constrained to lie on line L, is determined as follows. We pair the points arbitrarily in $\lfloor N_p/2 \rfloor$ pairs and compute the perpendicular bisectors of each pair. If any bisector is parallel to the line L then we drop the point that is nearer to L in the corresponding pair. In case the bisector is same as line L then any one of the points in the pair is dropped. After this, Subproblem 1 is solved for the median point, P, of the intersection points of L with the rest of bisectors. If P is the constrained centre then the computation is over, otherwise one point corresponding to each of these bisectors can be dropped by localising the constrained centre on L with respect to P. Thus it can be seen that at least $\lfloor N_p/4 \rfloor$ points are dropped in one iteration. We iterate until fewer than four points remain when any straight forward algorithm is applied. This is the technique of prune and search in which the size of input set is truncated by a constant fraction in each iteration. It is easily seen that it runs in linear time in the present case.

The solution of Subproblem 3 is obtained as follows. We pair the points in $\mathcal{P}$ and form their perpendicular bisectors. Let the median slope of non-vertical bisectors of these be s_m. The bisectors of slope strictly larger than s_m are paired with bisectors of slope strictly smaller than s_m. As it can be seen, this pairing may not cover all the bisectors, if the number of positive slope bisectors is not equal to the number of negative slope ones. Next, intersection points of each of these bisector pairs are computed.

First we localise the centre of min-circle with respect to a vertical line which divides the total of the vertical bisectors and the intersection points above in two halves. If the centre of min-circle lies on this vertical line then the min-circle is determined, otherwise let J_y be the halfplane, determined by this line, in which the centre of min-circle is localised. Next we localise the centre of min-circle with respect to a line with slope s_m such that it divides the total set of the intersection points in $\overline{J}_y$ and the bisectors with slope s_m in the above. As previously, if the centre of min-circle lies on this line then we terminate our computation, otherwise let J_m be the halfplane determined by this line in which the centre of min-circle is localised. Then we drop a point which is nearer to the centre of min-circle for each corresponding bisector that does not intersect $J_y \cap J_m$.

It can be easily shown that there are at least $\lfloor N_p/4 \rfloor$ bisectors which are one of these — the paired bisectors, the bisectors with median slope or the vertical bisectors. Thus, in every iteration at least $\lfloor N_p/16 \rfloor$ points are dropped. These steps are repeated with the truncated set of points until there are fewer than sixteen points when any straight forward algorithm to compute min-circle is applied. This algorithm runs in linear time.

As we will show later in the next section, the method of computation of intersection radius of a set of lines, $\mathcal{L}$, also uses the same principles.

2.4 Intersection Radius Problem for Lines, Rays and Line Segments in the Plane

In the previous section we presented a linear time algorithm to determine min-circle of a set of points $\mathcal{P}$. In this section we shall design an algorithm to determine the minimum stabbing disk, *min-disk* , which intersects a given set of line segments, rays and lines.

The real difficulty to apply the same technique to compute the intersection radius of line segments/rays is that the bisectors of these can be curved lines. The nature of these depend on the relative positions of the line segments/rays in the plane. These may contain parts of bisector of two endpoints (a straight line), parts of bisector of an endpoint and a supporting line (a parabola) and a part of bisector of the supporting lines of both (an orthogonal pair of straight lines). It is the presence of parabolas in the set that does not allow us to apply the pruning technique to these bisectors. However, the curved parts, namely parabolas, in the bisectors of line segments owe their existence to the bisectors of line and points only. So, we can alternatively look at a similar problem in order to circumvent this difficulty. This problem is — How can we determine the intersection radius of a mixed set of objects in the plane containing both lines and points. An added interest in this latter problem is that its solution is also used in the solution of the former problem.

Further, each line segment is considered to be equivalent to two oppositely directed rays having the endpoints of the line segments as their respective tails. Thus the intersection radius problem for a set of N_{ls} line segments can be reduced to the intersection radius problem for a set of $2N_{ls}$ rays in the plane. The latter problem is solved by combining the prune and search strategy of Megiddo [21] with the novel idea of replacing a ray by a line or a point in the pruning step. This is done by breaking up the problem into the following two subproblems:

Subproblem 4 *Given a set of lines and points in the plane, compute the smallest* *radius disk that intersects these.*

Subproblem 5 *Given a set of rays, points and lines in the plane, show how a fraction of the rays can be replaced by lines or points such that the intersection radius of the new set is the same as that of the original one.*

We will use the subscripts l, p and r to denote functions related to lines, points and rays, symbols L, P and R to denote a line, a point and a ray and denote finite sets of these by $\mathcal{L}, \mathcal{P}$ and $\mathcal{R}$, respectively. The functions $g_l(x,y), g_p(x,y)$ and $g_r(x,y)$ have the following definitions:

$$\begin{aligned} g_l(x,y) &= \max\left\{d_l(L,(x,y)) \;\middle|\; L \in \mathcal{L}\right\}; \\ g_p(x,y) &= \max\left\{d_p(P,(x,y)) \;\middle|\; P \in \mathcal{P}\right\}; \\ g_r(x,y) &= \max\left\{d_r(R,(x,y)) \;\middle|\; R \in \mathcal{R}\right\}, \end{aligned}$$

where $d_l(L,(x,y)), d_p(P,(x,y))$ and $d_r(R,(x,y))$ respectively denote the (Euclidean) distance of the point (x,y) from a line L, a point P and a ray R. We define $g(x,y)$ as

$$g(x,y) = \max\left\{g_l(x,y),\ g_p(x,y),\ g_r(x,y)\right\}.$$

It can be easily shown that the functions $g_l(x,y), g_p(x,y)$ and $g_r(x,y)$ are all convex so that $g(x,y)$ is also convex.

Let $\mathcal{S}$ be any finite set of lines, points and rays. The convexity of this last function enables us to answer two key questions about the minimum radius disk that intersects $\mathcal{S}$. The first is that if we constrain the centre of the minimum radius stabbing disk to lie on a fixed line L then to which side of a given point (α,β) on L does the constrained centre lie. The second is that given a line L to which side of it does the centre of the minimum radius stabbing disk lie. Without any loss of generality, we can take L to be the x-axis of an orthogonal frame of reference.

Let us answer the first question. Clearly, we can compute $g(\alpha,0)$ for the set S in $O(N_S)$ time, where N_S is the size of S. Let S' be the subset of objects whose distance from $(\alpha,0)$ is $g(\alpha,0)$ or, more simply, those that touch the disk. Since $g(\alpha,0)$ is convex, if the contact points of all the objects in S' lie to the left(right) of the vertical line through $(\alpha,0)$ then the centre of the constrained minimum radius stabbing disk lies to the left(right) of $(\alpha,0)$. Otherwise, $(\alpha,0)$ itself is the required centre.

The second question is also easily answered. We compute a minimum radius stabbing disk whose centre is constrained to lie on the line L. If the contact points of the objects in S' span an arc greater than or equal to a semi circle of the disk boundary, then the computed disk is the required minimum radius spanning disk. Otherwise, the centre lies in the same halfspace, determined by L, as in which the mid-point of the chord of the above spanning arc lies. This again follows from the convexity of the function $g(x,y)$.

We first present a linear time algorithm to determine a minimum radius stabbing disk that intersects a given set of points and lines. When the objects to be intersected are only points, the algorithm of Dyer or Megiddo [11, 21] for the 1-centre problem can be used. The situation is more complex when lines are also included in the set of objects to be intersected. Our algorithm uses the basic prune and search technique of Dyer or Megiddo, referred to above. We describe the technique for lines only. We will then argue at the end of section that the addition of points does not change the underlying concept.

2.4.1 Intersection Radius Problem for Points and Lines in the Plane

Here we first present a linear time algorithm to determine min-disk of a set of lines, $\mathcal{L}$. Let the size of $\mathcal{L}$ be N_l. The min-disk of $\mathcal{L}$ can be computed by transforming the problem to a linear programming problem and then solving it using any of the known methods [3]. The algorithm given here uses a different approach and applies the basic prune and search technique of Dyer and Megiddo [11, 21].

As in previous section, we progressively solve three similar problems. These problems are — First, computation of the stabbing disk of minimum radius when the centres of stabbing disks are at a fixed point P, computation of minimum radius stabbing disk when the centre of stabbing disks lie on a fixed line L and lastly, computation of the unconstrained min-disk.

The first of these problems is solved in linear time by computing the maximum of the distances from each line in $\mathcal{L}$ to point P. The required min-disk has this distance as its radius.

In the solution of the other two, as in the case of points, the corresponding bisectors of pairs of lines play an important role. These bisectors are either orthogonal pair of angular bisectors (for pair of non parallel lines) or parallel midway line (for pair of parallel lines). The lines are dropped in the computation of intersection radius by the application of the following rule:

If the centre of min-disk is localised in one of the regions, defined by the bisector(s) of two lines in the set $\mathcal{L}$, then the line that is nearer to the localised region can be dropped.

To prune away a constant fraction of lines in $\mathcal{L}$ by applying this rule we need to localise the centre of the min-disk with respect to the bisector(s) of a constant fraction of pairs of lines. The localisation of centre with respect to a single line is done by computing the function $g(x, y)$ as mentioned above. We discuss next how the localisation of the centre of min-disk, with respect to the bisectors (angular/parallel bisectors) of a constant fraction of pairs of lines in $\mathcal{L}$, in linear time, is done.

▮ *A Constrained Version of the Problem*

We discuss the problem of computing the min-disk for a set of lines, $\mathcal{L}$, such that the centre of this disk is constrained to lie on a line L, which is assumed here to be the x-axis of an orthogonal frame of reference. We also assume that $\mathcal{L}$ has at most two lines parallel to L, since any other lying between these two does not change the intersection radius. Furthermore, out of these two lines we can prune away the one that is closer to L. So there is no loss of generality in assuming that there is at most one line parallel to L. To the remaining $N_l'(\geq (N_l - 1))$ lines we apply the prune and search strategy in the following way.

We first identify a group of roughly half the line-pairs such that one of their bisectors does not intersect an interval on L which contains the centre of the constrained min-disk. This can be done in linear time as follows.

We compute the median x_m of the intersections of all the lines with L. Then we determine the value of $g(x_m, 0)$ at this point and use this to locate the constrained centre on L with respect to $(x_m, 0)$. If the constrained centre, denoted by $(x^*, 0)$, lies to the right (left) of this median point, the required half consists of those lines whose intersections lie to the left (right) of the median. We label the lines L_i, with $1 \leq i \leq \lfloor N_l'/2 \rfloor$. It will suffice to discuss the case in which the constrained centre lies to the right.

Consider the set of line-pairs (L_{2i-1}, L_{2i}), with $1 \leq i \leq \lfloor N_l'/4 \rfloor$.

For each parallel line-pair it is clear that we can prune away the one that is closer to x_m. The pruning mechanism is non-trivial for non-parallel pairs, since we have to do a finer location of the constrained centre. Of these, we can prune away the line closer to x_m if one of the angular bisectors is parallel to line L.

Each non-parallel pair (L_{2i-1}, L_{2i}) has an associated pair of angle bisectors. The intersection with L of one of these does not lie between the intersections of the lines themselves (Fig. 4). Let d_i be this intersection point.

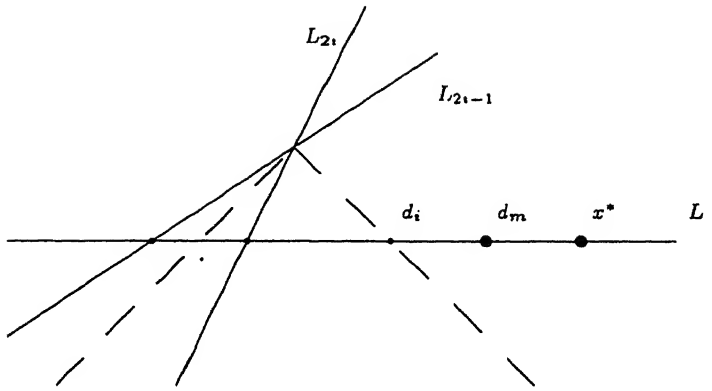

Figure 4: d_i does not lie between the intersections of lines L_{2i} and L_{2i-1}.
d_m lies to the left of x^*.

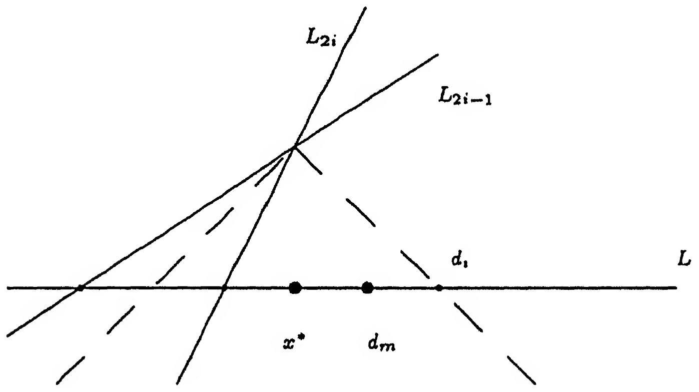

Figure 5: d_m lies to the right of x^*.

As before, we locate the constrained centre with respect to the median d_m of those intersections among these that lie to the right of x_m. Either $x^* = d_m$, or we have the following cases:

Case 1: $x^* < d_m$

Consider a d_i that lies to the right of d_m. Since x^* lies to the right of all the L_i's, the angle bisector that intersects L at d_i, its associated pair, together with the lines that these bisect give rise to the configuration of Fig. 5. Since x^* lies as shown we can prune away one of the lines of the pair, (L_{2i-1}, L_{2i}). It can be easily shown that we can prune roughly one-eighth of the lines we started with.

Case 2: $x^* > d_m$

It is easy to see that in this case also approximately one-eighth of the lines we started with can be thrown away (Fig. 4). ∎

The above pruning takes $O(N_l)$ time. We repeat this process until no more lines can be pruned, when the optimal solution can be obtained by some brute force method. It can be shown that no more than eight lines are left at this stage. If the initial set of lines contained a line parallel to L, the radius of the constrained min-disk is the maximum of the optimum value obtained and the distance from this parallel line to L. The total running time of this algorithm is easily seen to be $O(N_l)$.

If S also contains points, each pruning step is carried out in two substeps. In the first substep we prune points, followed by lines in the second or the other way round. To prune points, we first pair them arbitrarily, and compute the median point of the intersections of L with the bisectors of these pairs. Then we determine the min disk, D, centred at the median point, for the set S. Next, to determine on which side of the centre of D on L the constrained centre lies, we examine how the points of tangencies of lines of S, that touch the disk D, and the points of S, that lie on its circumference, are distributed with respect to the vertical line through the centre of D. When we prune lines next, again points of S are ignored similarly in the steps in which we determine x_m and d_m.

In summary, we note that when pruning objects of one kind, the objects of the other kind become transparent whenever we need to find a point on L to serve as the centre of stabbing disks for all the objects in the set currently under consideration.

If N_l and N_p be the number of lines and points in S respectively, it is easy to see that we prune away at least one-eighth of the total number of objects, viz $N_l + N_p$. We repeat this process until we cannot prune any more objects. The constrained centre

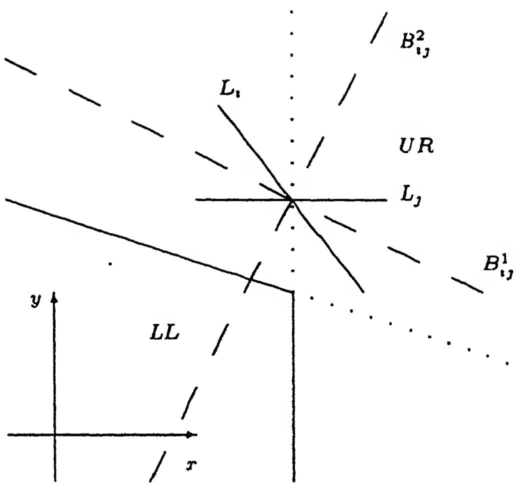

Figure 6: One of the bisectors of L_i and L_j (B^1_{ij} in the figure) does not intersect the interior of LL.

can then be determined by a brute force algorithm. The whole process requires linear time.

The Unconstrained Centre Problem

Assume that $\mathcal{L}$ is the set of lines in $\mathcal{S}$. We compute the min-disk of $\mathcal{L}$ in the unconstrained case. As before, we will indicate later how to handle the addition of points.

We pair up the lines in $\mathcal{L}$ arbitrarily and compute the angle bisectors of each pair. In the degenerate case of a pair of lines being parallel, the angle bisectors reduce to a single line parallel to and equidistant from the lines that make up the pair. When a pair of lines have distinct angle bisectors, these make up an associated pair.

The following observations are crucial. Given a pair of intersecting lines, if we can locate the region containing the centre in a quadrant defined by the angle bisectors of the pair then we can prune away one of the lines. For a pair of parallel lines we can do the same if we can locate this region in a halfplane determined by their "zero angle bisector". We indicate below how we do this for a fraction of such pairs.

Consider first the subset of vertical bisectors. We locate the unconstrained centre with respect to a median vertical bisector L_1. Assume that the centre of min-disk lies in the left halfspace J_1 of this median line. Clearly about half of the vertical bisectors do not intersect J_1. We now compute the median slope of the non-vertical bisectors and pair up arbitrarily in this subset a bisector which has slope greater than the median slope with one which has a smaller slope. The bisectors in each of these pairs necessarily intersect as they have unequal slopes. Next we compute the median of all the unpaired bisectors whose slopes are equal to the median slope. We again locate the unconstrained centre with respect to this median bisector L_2 which has median slope. Assume that the centre lies in the halfplane J_2 below this line. Clearly about half of the median slope bisectors that lie above L_2 do not intersect J_2. The centre now lies in $J_1 \cap J_2$. To be able to prune any lines, we need to refine the location of the centre still further as in the worst case, we may neither have any vertical bisectors nor any whose slopes are equal to the median slope. Therefore, we do the following with respect to the above pairs of intersecting bisectors.

We first locate the unconstrained centre with respect to a vertical line through the median of the x-coordinates of their intersections. We can assume, without any loss of generality, that the unconstrained centre lies to the left of this line. We now project, parallel to the median slope, the intersection points that lie to right of this line onto the y-axis. We then locate the unconstrained centre with respect to a line, parallel to the median slope, passing through the median of these projections. Again there would be no loss in generality if we assume that the unconstrained centre lies below this line so that now it lies in the lower left quadrant LL, determined by this line and the earlier one.

This ensures that at least a fourth of the pairs of bisectors in this class have their intersections in the upper right quadrant UR. Consider one such pair. Since the bisector with slope smaller than the median slope does not intersect LL, at least one-eighth of the bisectors whose slopes are not equal to the median slope do not intersect LL. We note that this argument does not depend upon which quadrant the unconstrained centre lies in. Thus, we have a set of bisectors which do not intersect

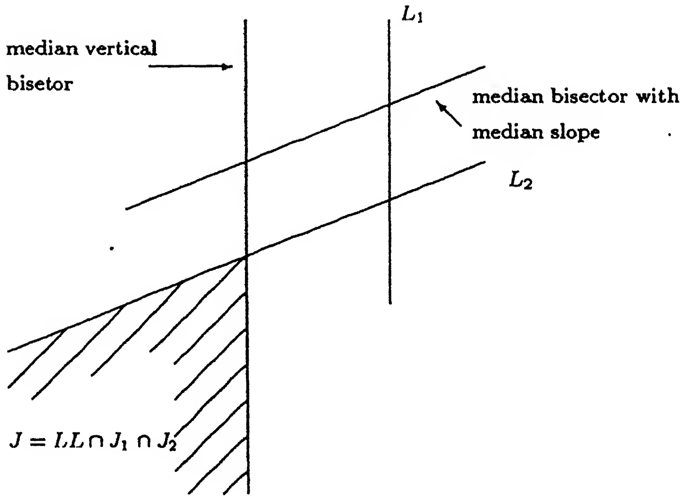

Figure 7: The region $J = LL \cap J_1 \cap J_2$.

the region $J = LL \cap J_1 \cap J_2$ in which the centre of min-disk lies (Fig 7). In the worst case even this set of bisectors does not enable us to prune any lines because there may be no associated pairs of bisectors or bisectors due to a pair of parallel lines in this set. Further, in the worst case, it may happen that all of the associated bisectors of this bunch intersect J.

We need to do one more final refinement of the location of the unconstrained centre with respect to these associated bisectors that intersect J. If we repeat the above steps with the above associated set of bisectors we get a region J', containing the centre and a fraction of these bisectors which do not intersect it. Each of these, together with its associated pair from the earlier set of bisectors we found that do not intersect J, contain in one of its quadrants the region $J \cap J'$. Region $J \cap J'$ contains the unconstrained centre. Therefore one of the lines whose angle bisectors these are can be pruned. At the same time we can prune away one of the lines of each parallel pair whose "zero angle bisector" does not intersect $J \cap J'$.

Calculation shows that at least $\lfloor N_l/64 \rfloor$ of the lines are pruned away.

We repeat this process until no more lines can be discarded. It can be shown that there are no more than 64 lines in $\mathcal{L}$ at this stage and we use some brute force

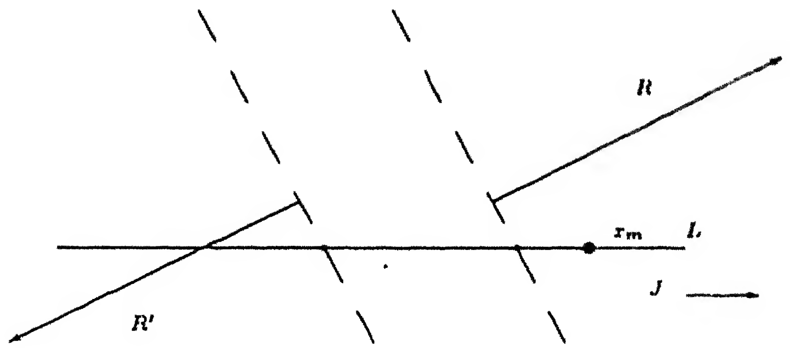

Figure 8: Ray R is replaced by the supporting line of R; R' is replaced by its tail point

method to compute the unconstrained centre.

When points are also included in the set, $\mathcal{S}$, of objects to be intersected, we go about the pruning step in exactly the same way as in the constrained case. In each such step we throw away a well determined fraction of the points and the lines. Repeating this process we get a linear time algorithm for the unconstrained centre problem for $\mathcal{S}$.

2.4.2 Intersection radius for rays

The constrained problem for a set $\mathcal{R}$ of N_r rays in the plane can be reduced to the problem of pruning for a set of rays, lines and points as detailed below.

For each ray $R_i \in \mathcal{R}$, consider the line L_i normal to it and passing through its tail. We compute the median x_m of the intersections of these normals with the constraint line L and locate to which side of this median on L the constrained centre lies. Suppose the constrained centre lies to the right of this median. Then for each normal which intersects L to the left of this median point we replace the corresponding ray by a line or point according to the following replacement rule :

If the ray and the median point lie in the same halfspace of the two halfspaces determined by the normal then the ray is replaced by the line

> *which contains the ray. Otherwise, the ray is replaced by its tail point (Fig. 8).*

It is important to note that neither of these replacements changes the radius of the relevant stabbing disks of the original set of rays. Thus, if N_r be the number of rays in $\mathcal{R}$, at least $\lfloor N_r/2 \rfloor$ rays are replaced by either a point or a line and our new set of objects consists therefore of lines, points and rays. Next, all the points and lines are considered for pruning as described in the § 2.4.1. We thus discard a fraction of the rays from further consideration. These two substeps are iterated until no more objects can be discarded, when any brute force method can be applied to compute the min-disk centred on L. It is easy to check that the algorithm runs in linear time.

To solve the unconstrained version of the intersection radius problem for a set of rays, we need to replace a fraction of the rays by points or lines in linear time. This can be done as follows.

As in the constrained case, we start with the normals through the tails of the rays. Proceeding identically as in the case of the unconstrained problem for lines, we determine a region J which contains the unconstrained centre and is not intersected by at least one-eighth of these normals. We do not need to iterate twice as in the case of lines. Only one iteration suffices for the replacement of a fraction of rays. The ray corresponding to each of these normals can therefore be replaced by a line or a point according to a similar replacement rule.

> *If the ray and localised region J lie in the same halfspace of the two halfspaces determined by the normal then the ray is replaced by the line which contains the ray. Otherwise, the ray is replaced by its tail point.*

Thus in a single iteration we replace about one-eighth of the rays by lines or points. Next we use the two-step pruning process on the set of lines and points generated so far to throw away a fraction of them. Repeating these two substeps on the modified set of rays, lines and points, and then applying any brute force method when no pruning takes place we get a linear time algorithm for intersection problem of line segments in the plane.

2.5 Concluding Remarks

In this chapter we have described optimal algorithms for computing the smallest radius disk which intersects a set of line segments in the plane using a novel approach. It would be worth investigating whether a similar approach can be used for other kinds of stabbing problems.

Chapter 3

An Optimal Algorithm for the Intersection Radius of a Set of Convex Polygons

In the last chapter it was shown how the minimum stabbing disks and intersection radii for finite sets of points, lines, rays and line segments can be computed. These were computed in linear time by combining the prune and search strategy of Megiddo [21] with the strategy of replacing line segments and rays by lines or points [3]. In this chapter, we enlarge the scope of this technique by showing that it can also be used to compute the intersection radius of a finite set of convex polygons in linear time. Moreover, it is immaterial if the set also contains other types of geometric objects such as points, lines, rays, line segments, half planes and wedges. In fact, we will show how to handle such a mixed set of objects in a unified way.

The bisectors of line segments, as we have seen in the previous chapter, are made up of lines and (parabolic) curves. Here too the bisectors of convex polygonal disks are formed by not only parabolic curves but also regions of non-zero width. Such a case arises when a pair of convex polygonal disks have non-empty intersection and we have these regions of non zero finite area included in the bisector(s) of these. We present a method in this chapter in which this non linearity of bisectors does not pose any problem and we compute the minimum stabbing disk of a finite set of convex polygonal disks in linear time. But before doing this we first formalise the notions of replacement, substitution and localisation. These concepts were implied when we used these in their specific contexts in the previous chapter but nevertheless, were not discussed there.

3.1 Introduction

Let C be a finite collection of objects in the Euclidean plane. The *stabbing* problem consists of computing an object (the stabber) which intersects each member of C. Typically, the stabber could be a line, a disk, a line segment etc. and C could be a collection of points, lines, line segments, rays, circles, polygons or any mix of these. A survey of some recent results is available in the paper by Houle et al. [15].

The *intersection radius* of a finite collection of geometrical objects is the radius of the smallest closed ball that intersects all the objects in the collection. A classical problem in this area is that of finding the intersection radius of a finite set of points in the plane, which is also known as the 1-centre problem [30, 31, 6]. The corresponding disk is called the minimum stabbing disk. It was shown by Megiddo, and subsequently by Dyer [22, 12], how this can be solved in linear time. More recently, Welzl [34] has given a randomised algorithm for this problem which runs in expected linear time. However, until an earlier paper by Bhattacharya et al. [3] no attempt was made to extend this to a more complex collection of objects than points or to a collection containing different kinds of objects.

New attempts have been made recently to find more complicated stabbers for the stabbing problem, or to find the best stabber which optimises some measure defined on the class of stabbers in question. Goodrich and Snoeyink [14] presented an $O(n \log n)$ algorithm to find a convex polygon whose boundary intersects each of n parallel line segments. Rappaport and Meijer [24] showed that a perimeter minimising convex polygon that intersects each of n parallel line segments can be found in $O(n \log n)$ time. They have also extended their result to a set of isothetic line segments. Mukhopadhyay et al. [27] have shown that for a set of parallel line segments an area minimising convex polygonal disk can also be found in $O(n \log n)$ time. Bhattacharya et al. [2] gave an $O(n \log n)$ algorithm for computing the shortest line segment that intersects a set of n line segments in the plane.

Bhattacharya et al.[3] showed that when $\mathcal{C}$ is a collection of line segments the intersection radius can be found by combining the prune and search strategy of Megiddo [21] with the strategy of replacing line segments with points or lines. This was discussed in the last chapter. In this chapter we enlarge the scope of this technique by showing that the intersection radius can also be found in linear time when $\mathcal{C}$ is a collection of convex polygons. Really, it is immaterial if $\mathcal{C}$ also contains other geometric objects like lines, points, rays etc. We show how it is possible to treat such a mixed collection of objects in a unified way.

The organisation of this chapter is as follows. Section 3.2 contains the necessary geometric and algorithmic preliminaries. In Section 3.3 we describe the algorithm and analyse it in the following section. The last section contains conclusions and directions for further research.

3.2 Preliminaries

In the rest of this chapter we will adopt the following notation scheme: collections of objects will be denoted by letters in script style such as $\mathcal{C}, \mathcal{L}, \mathcal{P}, \ldots$, objects by capital letters such as $L, P, R, \ldots$, and points by small letters $p, q, r, \ldots$ Let the size of a collection $\mathcal{C}$ be denoted by $N_{\mathcal{C}}$.

The algorithm that we shall describe in the following section is based on three important techniques. These are: *replacement* of a complex geometrical object by a set of simpler objects, *localisation* of the centre of minimum stabbing disk in a more restricted region and thereafter *filtration* of some of the objects which do not play a role in determining the centre of the minimum stabbing disk. Together, the latter two make up the prune and search technique that was first formalised by Megiddo [21].

We discuss these strategies, *replacement*, *localisation* and *filtration*, in detail in the next three sections.

Let $\mathcal{C}$ be a collection of n objects in the plane where each object is either a point, a line, a line segment, a ray, a wedge or a convex polygonal disk. The problem is to determine the minimum stabbing disk, min-disk, of the collection $\mathcal{C}$.

In the following discussion J is a fixed subset of plane containing the centre of min-disk. In the general case, J is initially the whole plane, and gets smaller as the computation proceeds.

In the constrained case, J is always a subset of the line on which the centres of stabbing disks are constrained to lie. All the discussion in this section is also applicable to the constrained case where J is such restricted to a line.

Let C and $\mathcal{C}$ denote an object and a collection of objects respectively. The distance of a point p from C, $d(p,C)$, is the shortest Euclidean distance of p from C. The distance of p from $\mathcal{C}$, $d(p,\mathcal{C})$, however, is the largest of all distances from p to the objects in $\mathcal{C}$. The *stabbing radius*, SR(J,$\mathcal{C}$), where the centre of the stabbing disks are constrained to lie in J, is the minimum of the distances from points in J to $\mathcal{C}$. The *intersection radius*, IR($\mathcal{C}$), is the unconstrained stabbing radius. In summary, we have

$$\begin{aligned} d(p,C) &= \inf_{q\in C} d(p,q), \\ d(p,\mathcal{C}) &= \max_{C\in\mathcal{C}} d(p,C), \\ \mathrm{SR}(J,\mathcal{C}) &= \min_{p\in J} d(p,\mathcal{C}), \end{aligned}$$

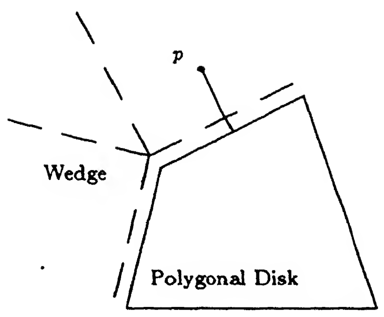

Figure 9: A polygonal disk, and its substitution by wedges (one of the wedges seen slightly shifted)

$$\mathrm{IR}(\mathcal{C}) \;=\; \mathrm{SR}(\Re^2, \mathcal{C}),$$

where $d(p,q)$ is the distance from p to q.

The *distance bisector* of two objects S and T is defined as the set of all points for which the distance to S is equal to the distance to T, i.e., it is the set

$$\left\{\, p \mid d(p,S) = d(p,T),\ p \in \Re^2 \right\}.$$

3.2.1 Object Replacement

To simplify the computation it is helpful to divide a convex polygonal disk into a set of elementary parts such that the distance function is not modified. This allows us treat these in a uniform manner in the algorithm. To see how we can divide a convex polygonal disk, let us first see how the distance from a point p in $\Re^2$ to a convex polygonal disk is computed. This distance is either equal to the distance from p to some vertex, or it is equal to the perpendicular distance to some side, or it is zero when the point lies inside the disk. A *wedge* is the non-reflex region bounded by its two infinite sides. It can be easily seen therefore that the distance from p to this disk is the maximum of the distances to the wedges formed by taking the vertices and their adjacent sides (Fig 9). So if a convex polygonal disk in $\mathcal{C}$ is substituted by these wedges then the intersection radius of the resulting collection does not change.

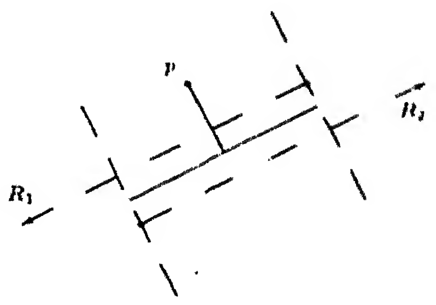

Figure 10: A line segment, and its substitution by two rays (seen shifted)

Each line segment can be similarly replaced by two oppositely directed rays whose intersection is the line segment itself (Fig 10). The distance of the line segment from a point p in $\Re^2$ is either equal to the distance of the endpoints or it is equal to the perpendicular distance to the supporting line; this distance is clearly equal to the maximum of the distances of p from the two rays.

These examples motivate the following theorem.

Theorem 3.1 *Let C be an object and $\mathcal{D}$ a collection of objects such that $d(p,C) = d(p,\mathcal{D})$ for all $p \in J$. Then for any collection $\mathcal{C}$,*

$$\mathrm{SR}(J,\mathcal{C} \cup \{C\}) = \mathrm{SR}(J,\mathcal{C} \cup \mathcal{D}).$$

Proof: From the definitions of intersection radius and distance functions, we have

$$\begin{aligned}
\mathrm{SR}(J,\mathcal{C} \cup \{C\}) &= \min_{p \in J} \max\{d(p,\mathcal{C}), d(p,C)\} \\
&= \min_{p \in J} \max\{d(p,\mathcal{C}), d(p,\mathcal{D})\} \\
&= \min_{p \in J} d(p,\mathcal{C} \cup \mathcal{D}) \\
&= \mathrm{SR}(J,\mathcal{C} \cup \mathcal{D}).
\end{aligned}$$

This proves the theorem. ∎

When C is a line segment, a ray, a convex polygonal disk, a halfplane or a wedge, we can apply Theorem 3.1 to replace C by some simpler object(s).

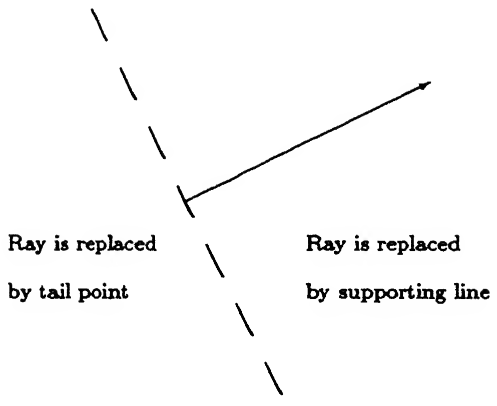

Figure 11: A ray R, its associated normal and regions

In the two examples mentioned above J is the whole plane. This allows unconditional replacement of line segments and convex polygonal disks by rays and wedges, respectively. Let us take another set of examples, when J is only a proper subset of the plane, C is a ray, a halfplane or a wedge, and the replacement is done conditionally. Let R be a ray in C. The normal to this ray through its tail point divides the plane in two halfplanes. If we localise the centre of min-disk in one of these halfplanes then the distance from any point of the localised region to the ray is equal to either the distance to the tail point or the perpendicular distance to the supporting line of the ray R (Fig. 11). This means that we can replace R either by its tail point or its supporting line.

Similarly if J is localised in the interior of a halfplane, H, in C then the distance from any point of J to H is zero, otherwise if J is localised in the interior of $\overline{H}$ then the distance from any point of J to H is equal to the distance to the boundary of H. We can respectively discard H or replace H by its boundary line in C if these cases arise (Fig 12).

Let us draw an outward normal to each of the sides at the apex of a wedge W. The whole plane is then divided into four (unequal) quadrants by these normals and sides of W. If we somehow localise the region J in one of these quadrants then the distance of the points in J from the wedge has one of the following values: 0, when J is localised inside the wedge; perpendicular distance to one of the sides, when

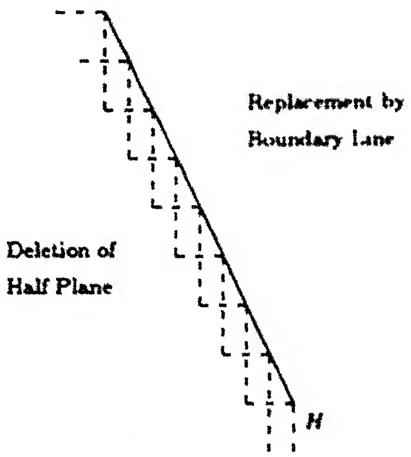

Figure 12: A halfplane H, its associated boundary line and regions

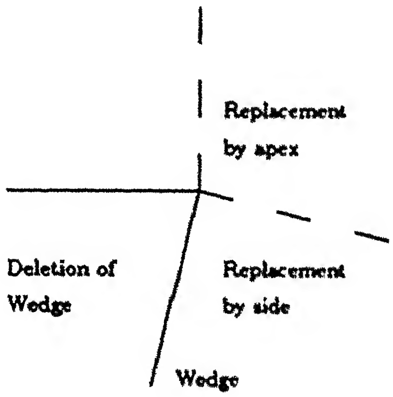

Figure 13: A wedge W, its associated normals and regions

J is localised in the region bounded by a side and the normal to it; or, distance to the apex, when J is localised in the region bounded by two normals. We can respectively discard W, replace W by the relevant side, or replace W by the apex, in these cases(Fig 13).

We shall henceforth view a convex m-gon as the collection of m wedges defined by the vertices and the sides incident on them. Our problem, therefore, is equivalent to that of computing a min-disk for a set of $N_W(=\sum m$, sum taken over all the convex polygons) wedges. Likewise, we shall view a line segment as a pair of oppositely directed rays defined by its end points [3].

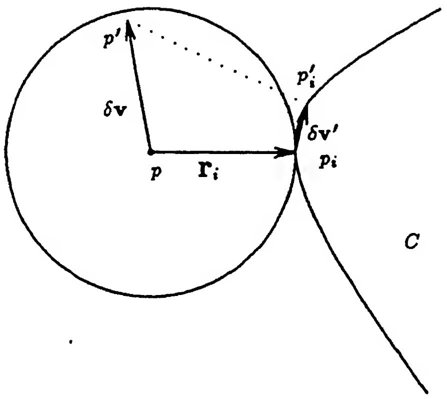

Figure 14: Stabbing disk is a min-disk

3.2.2 Localisation of the Centre of Min-Disk

In the discussion below we assume that the objects are convex sets of points so that all distance functions are convex.

An object C is said to touch a disk of radius r centred at point p if $r = d(p, C)$, and the vectors from p to the points of contact are said to be its contact vectors. The following theorem characterises min-disk.

Theorem 3.2 *A stabbing disk of radius r centred at point p is min-disk iff all of its k contact vectors, $\mathbf{r}_i$, with $1 \leq i \leq k$, are linearly dependent satisfying*

$$\sum_{i=1}^{k} \lambda_i \mathbf{r}_i = \mathbf{0},$$

where $\lambda_i \geq 0$, for $1 \leq i \leq k$, with some $\lambda_i \neq 0$ and $k \geq 2$.

Proof: The proof makes use of the behavior of the distance function in the neighbourhood of p, which depends only on the contact vectors of the stabbing disk.

Let the contact vectors $\mathbf{r}_i$ satisfy the relationship given in the theorem. Then for any arbitrary vector $\mathbf{v}$, we have

$$\sum_{i \leq k} \lambda_i (\mathbf{r}_i \cdot \mathbf{v}) = 0$$

and hence, $(\mathbf{r}_i \cdot \mathbf{v}) \leq 0$ for some i, with $1 \leq i \leq k$. Let this object be C. Now, if p is displaced to p' by a small amount $\delta\mathbf{v}$ in the direction of $\mathbf{v}$ and if $\delta\mathbf{v}'$ is the corresponding displacement of the new contact point, p'_i, of the object C (Fig 14) then the square of the new distance, d'_i, of the new centre p' from the object C is

$$\begin{aligned} d_i'^2 &= |\mathbf{r}_i - (\delta\mathbf{v} - \delta\mathbf{v}')|^2 \\ &= |\mathbf{r}_i|^2 + |\delta\mathbf{v} - \delta\mathbf{v}'|^2 - 2(\mathbf{r}_i \cdot \delta\mathbf{v}) + 2(\mathbf{r}_i \cdot \delta\mathbf{v}') \\ &\geq |\mathbf{r}_i|^2 \\ &\geq r^2. \end{aligned}$$

Since C is convex and $\mathbf{r}_i$ is normal to it, $\mathbf{r}_i \cdot \delta\mathbf{v}'$ is non-negative and thus justifies the inequality above. Since $d(p,C)$ is the maximum of all distances from p to the objects in the collection C, $d(.,C)$ will also increase in the direction $\mathbf{v}$. Thus

$$\begin{aligned} d(p',C) &\geq d'_i \\ &\geq r \\ &\geq d(p,C). \end{aligned}$$

Hence, as $\mathbf{v}$ is an arbitrary vector, p is a local minimum of the function $d(.,C)$. Since $d(.,C)$ is convex, p is a global minimum too.

For the proof of the converse, we assume that the said disk is the min-disk. Then we prove that the centre p lies in the relative interior of the convex hull of the contact points.

We will prove this by contradiction. We assume the contrary that the centre of min-disk lies does not lie inside the convex hull of contact points. Then we can compute a line which separates the point p and the convex hull. Let $\mathbf{v}$ be the normal vector to this line, contained in the same halfplane as the convex hull. Then $\mathbf{v} \cdot \mathbf{r}_i > 0$ for all i, with $1 \leq i \leq k$. From this it can be seen that $d(p,C)$ strictly decreases along $\mathbf{v}$, since

$$d(p',C)^2 \leq d(p',p_i)^2$$

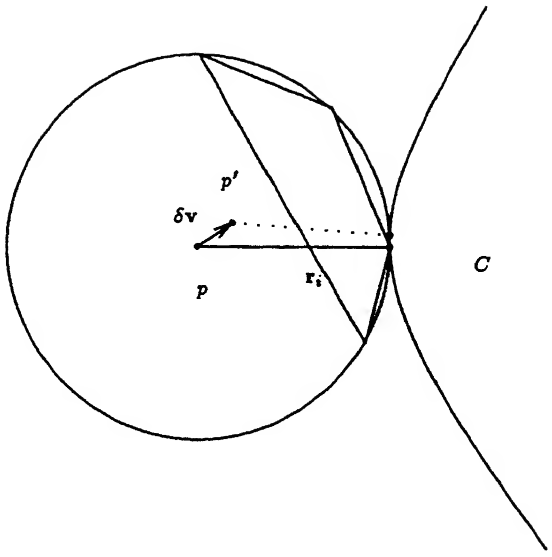

Figure 15: Centre of Min-Disk lies inside convex hull of contact points

$$\begin{aligned}
&\le |\mathbf{r}_i - \delta\mathbf{v}|^2 \\
&\le |\mathbf{r}_i|^2 - \delta\mathbf{v}\cdot(2\mathbf{r}_i - \delta\mathbf{v}) \\
&\le |\mathbf{r}_i|^2 \\
&\le r_i^2 \\
&\le d(p,C)^2.
\end{aligned}$$

This means that the radius of the min-disk is not minimum. Since this cannot be, therefore the assumption that the centre of the min-disk lies outside the convex hull of contact points is incorrect.

Let the vectors to p and the contact points be $\mathbf{p}$ and $\mathbf{p}_i$, with $1 \le i \le k$, respectively. Now, an interior point of a convex hull can be written as a positive linear combination of the extreme points. Thus

$$\mathbf{p} = \sum_{i\le k} \lambda_i \mathbf{p}_i,$$

$$\sum_{i\le k} \lambda_i = 1,$$

where $\lambda_i \geq 0$, with $1 \leq i \leq k$. Hence,

$$\sum_{i \leq k} \lambda_i(\mathbf{p}_i - \mathbf{p}) = 0$$

and therefore

$$\sum_{i \leq k} \lambda_i \mathbf{r}_i = 0.$$

∎

The above theorem is also applicable in the constrained case with a slight modification that here the contact vextors projected onto the constraint line are positive linearly dependent. Once we have computed a stabbing disk, if it is not a minimum one as determined by Theorem 3.2, we can further localise the region in which the centre of min-disk lies with the help of the theorem below.

Theorem 3.3 *Given a stabbing disk of C centred at p, with k contact vectors $\mathbf{r}_i$, with $1 \leq i \leq k$, the centre of min-disk lies in the set J' given by*

$$J' = J \bigcap \left\{ \bigcap_{i \leq k} H_i \right\},$$

where H_i is the halfspace, normal to $\mathbf{r}_i$, that passes through p and contains the contact point and the tangential object. Furthermore,

$$\mathrm{SR}(J, C) = \mathrm{SR}(J', C).$$

Proof: Let the radius of stabbing disk be r. We have to show that centre of min-disk lies in every H_i. Let the corresponding object to H_i be C. The radius of min-disk has to be smaller than or equal to r. Therefore the centre of min-disk is at a distance smaller than or equal to r from every object in C and in particular from C. The feasible region of this centre will thus be a subset of H_i (because C

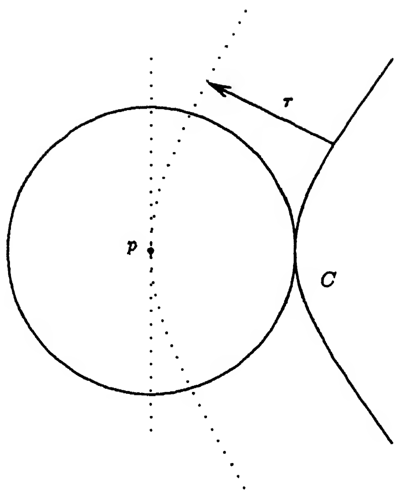

Figure 16: Localisation of Centre of Min-Disk

is convex). Every point not in H_i is at a distance larger than r from the object C (Fig 16). This proves the theorem. ∎

We can use this theorem to localise the centre of min-disk with respect to any arbitrary fixed line, L, in the plane. We first compute the minimum stabbing disk such that its centre is constrained to lie on L and then compute the region given by the expression of the theorem with this disk as the reference. Since the region contains only those points where the value of stabbing radius is smaller than the radius of given disk, therefore it necessarily does not contain any point of L. Thus we get a region that is fully contained in one of the halfplanes determined by L. Further, if we localise the centre of min-disk to a region J' we then onwards replace and filter objects with respect to newly located region J' and not the earlier located region J.

3.2.3 Pruning or Filtering Objects in $\mathcal{C}$

We now need a suitable criterion to discard objects from the collection $\mathcal{C}$ that are irrelevant in determining the centre of min-disk. The following theorem provides

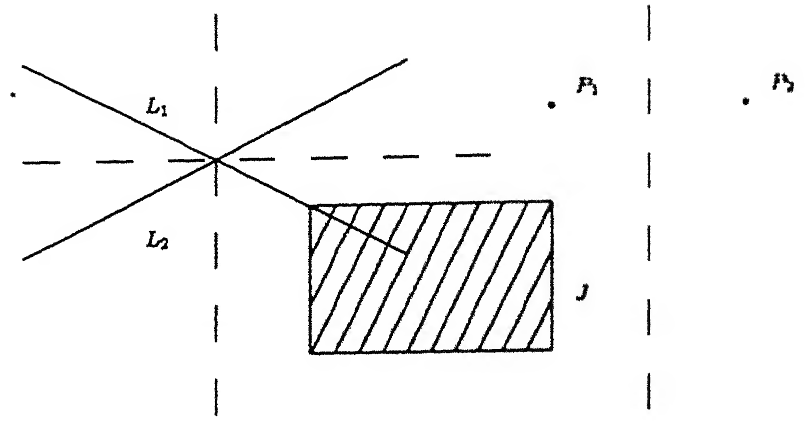

Figure 17: Filtering lines and points

such a criterion.

Theorem 3.4 *Let there be a pair of objects in $\mathcal{C}$ such that region J is contained in one of the regions determined by their distance bisector(s). Then one of these objects, whichever is nearer to J, can be discarded from $\mathcal{C}$ without affecting the intersection radius.*

Proof: From the definition of the distance bisector, if the region in J is contained in one of the regions defined by the bisector then, for every point p in J, distance of p from one of the objects is always smaller than the distance from the other object. Let this nearer object be C. Since the intersection radius is the maximum of the distances of objects of $\mathcal{C}$ over the points of the region J, the object C , its distance being smaller than the distance of the other object in the pair, does not play a role in the determination of min-disk. Hence, the object C can be deleted from the collection $\mathcal{C}$. ∎

For example, refer to Fig. 17 above. The angular bisectors of two lines L_1 and L_2 and the perpendicular bisector of two points P_1 and P_2 are shown. The nearer of the lines, L_1, is deleted because region J (the shaded region in the figure) is contained in the one of the four quadrants defined by the angular bisectors of L_1 and L_2.

Similarly, the nearer of the points, P_1, is deleted because the region J is contained in one of the halfplanes defined by the bisector of P_1 and P_2.

3.3 Intersection Radius Problem for Convex Polygons

In this section we present a linear time algorithm for computing the min-disk which intersects a collection C of convex polygons. In fact, we will be more general and assume that C also contains points, lines, rays, line segments, halfplanes and wedges. Since each line segment can be considered to be equivalent to two oppositely directed rays having the end points of the line segments as their respective tails, and a convex polygon as a collection of as many wedges as vertices (refer § 3.2.1), we will assume that C consists of lines, points, rays, halfplanes and wedges only.

We first solve a constrained version of the problem and then the unconstrained one. The motivation behind this is as follows. The intersection radius is the minimum of all stabbing radii over the domain of the plane. This problem can be formulated as a non-linear programming problem. The usual method of solution of such a problem in E^d consists of locating the optimal point with respect to a hyperplane in one dimension lower, by first solving the problem restricted to this hyperplane, and then locating the optimal point by computing the gradient of the minimising function.

Furthermore, to replace or filter objects, we need to locate the centre of min-disk with respect to either bisectors of pairs of points, bisectors of pairs of lines, the wedges or the normals to rays. So we apply the same technique, as in non-linear programming, to localise the region containing the centre. We note that we need not do this for every object in C in a single step. It suffices to do this only for a fraction of these objects. So, we successively localise the centre of min-disk with

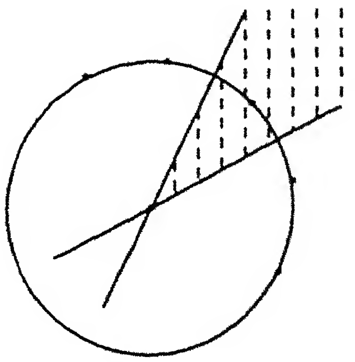

Figure 18: Localisation of Centre of Min-Disk in Linear Time

respect to a group of lines, which are chosen in such a way that there is always a fraction of the objects in C which are either discarded or replaced.

First we provide an efficient linear time implementation of the construction suggested by Theorem 3.3. Then we solve the constrained and unconstrained cases of the intersection radius problem in successive subsections.

3.3.1 An Efficient Implementation of Theorem 3.3

Suppose we are given a stabbing disk centred at c. We have to locate the centre of min-disk in a region, J, such that c lies on its boundary. We do this in the following way.

If the stabbing disk does not touch any object then its radius can be shrunk until it touches at least one object. This can be done in linear time by computing the maximum of the distance from c to the objects in C. Suppose there are k contact vectors of the stabbing disk with $k \geq 1$. If we compute the intersection of k halfplanes as in Theorem 3.3 by constructing the convex hull in the dual plane, then we will need $O(k \log k)$ time. This can take $O(N_C \log N_C)$ time in the worst case. In our application, however, this intersection can be computed in linear time because all these halfplanes have point c on their respective boundaries. Computation of their intersection is same as the convex hull computation in the dual plane of a set of collinear points. We only need to compute the two extreme halfplanes and the optimal centre lies in the intersection of these two halfplanes which is a wedge (Fig 18). If the intersection of the halfplanes is a single point, namely c, then c is the centre of min-disk. In the constrained case we need to locate the centre of minimum stabbing disk on a line L. For this we compute the part of the line L that lies inside this wedge.

3.3.2 The Constrained Centre Problem

In the constrained version of the problem we determine a min-disk whose centre is constrained to lie on a line L. Let $\mathcal{P}$, $\mathcal{L}$, $\mathcal{R}$, $\mathcal{H}$ and $\mathcal{W}$ be disjoint subsets of $\mathcal{C}$ containing points, lines, rays, halfplanes and wedges respectively such that their union is $\mathcal{C}$. We show how we can process points, lines, rays, halfplanes and wedges separately. We then unify these into a single iteration of the algorithm.

Points: We pair up the points in $\mathcal{P}$ arbitrarily and compute the median of the intersections of their bisectors with the line L. The centre is then localised with respect to this point on one of the half lines. We filter out at least one fourth of the points corresponding to the bisectors that do not intersect the localised region. We repeat this step until no filtering takes place.

Lines: We pair up the lines in $\mathcal{L}$ arbitrarily and compute their distance bisectors (angular bisectors for non-parallel lines, and mid-way line for parallel lines). We divide the intersections of these bisectors with the line L equally into four intervals on L. The centre of min-disk is localised in one of the intervals by doing a binary search on the boundary points. We filter at least one fourth of the lines corresponding to the distance bisectors that do not intersect the localised region. We repeat this step until no filtering takes place.

With rays, halfplanes and wedges we proceed in a slightly different way.

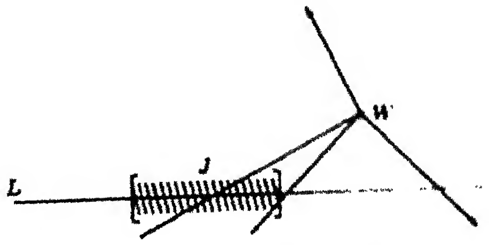

Figure 19: Intersection of objects with J in the constrained problem

Rays: We draw normals at the tail points of the rays in $\mathcal{R}$ and compute the median of the intersections of these normals with the line L. The centre of min disk is then localised with respect to this point. We can replace a ray, whose normal does not intersect the localised region, either by its tail point or its supporting line. We replace at least one half of the rays in each iteration. We repeat this step until no replacement takes place.

Halfplanes: We localise the centre of min-disk on the line L with respect to the median of the intersection of line L with the boundaries of halfplanes in $\mathcal{H}$. We replace a halfplane in $\mathcal{H}$ by its boundary line or discard it if its boundary line does not intersect the localised region. We replace/filter at least one half of the halfplanes in each iteration. We iterate until no filtration/replacement is done.

Wedges: For each wedge in $\mathcal{W}$, we draw outward normals to the sides at its apex. We divide the intersection(s) of the normals and the sides with the line L equally into four intervals of L. We then localise the centre of min-disk to one of these intervals. All those wedges (at least one fourth) whose sides and normals do not intersect the localised region are either filtered out or replaced by a line or point. We iterate until there is no replacement or deletion.

It is easy to design an $O(N_C \log N_C)$ algorithm, based on the above facts. We first convert all the rays in C to lines or points, then we filter or replace all the halfplanes and wedges in C, and finally compute the intersection radius of a collection containing only lines and points. To obtain a linear time algorithm, we have to treat the objects in a more unified way in each iteration of the algorithm.

The basic idea is as follows: The derived lines obtained as above from different objects of C are classified as either active or inactive, depending on the fact, whether region J which contains the centre of min-disk is localised with respect to these. Further, active lines among these are assigned following weights: derived lines of wedges with four and three intersections with J – 16; derived lines of wedges and line pairs with two intersections with J – 18; derived lines of rays and line-pairs with one intersection with J – 27; and derived lines of point-pairs with one intersection with J – 36. At the beginning of computation all the derived lines that intersect L are active. In each iteration we first localise the centre of min-disk on L with respect to the weighted median of intersections of line L with the active derived lines. The locating of the centre of min-disk is done by invoking Theorem 3.2 and Theorem 3.3 to determine on which side of the weighted median intersection point the constrained centre lies. Finally, we replace and filter those objects, none of whose derived lines are active, by invoking Theorem 3.1 and Theorem 3.4. We repeat this until no active derived lines are made inactive.

It can be easily seen that in every iteration half of the active lines of about half the total weight are inactivated. New derived lines of at most only three-fourth of this discarded weight are added. So there in net reduction of weight by a constant fraction. Hence the algorithm is linear in total weight.

The details are given in the algorithm below. $\mathcal{D}$ is the active derived set of lines.

Algorithm 1 CONSTRAINED-CENTRE

```
Input: Bisectors of point-pairs and line-pairs, normals to rays,
  boundary lines of halfplanes and sides of and normals to wedges.
Output: Centre of constrained min-disk
begin
    $\mathcal{D}$ ⟵input set
    do
        Compute the weighted median intersection, $P$, of $\mathcal{D}$ with $L$
        Locate $J$ with respect to $P$ on $L$ that contains the
```

```
            constrained centre
          Update $\mathcal{D}$
          Filter and replace relevant objects of the collection $\mathcal{C}$
          Update $\mathcal{L}$
      while $\mathcal{D}$ is updated
      enddo
      Determine the constrained centre by some brute force method
end   ∎
```

The proof of correctness of the algorithm CONSTRAINED-CENTRE is a direct consequence of the theorems in Section 3.2.

3.3.3 The Unconstrained Centre Problem

The tasks of filtration and replacement are more involved in the unconstrained case. We first describe a method by which we can localise the region that contains the centre of min-disk which does not intersects a finite fraction of a given set of lines. As in the constrained case, we then show how we can process points, lines, rays, halfplanes and wedges separately. Finally, we unify these separate steps into a single iteration of the algorithm.

Let LOCALISE be the procedure which, given a set of lines $\mathcal{L}$ with integral weights, determines a region J that contains the centre and is disjoint from at least one eighth of the total weight of lines. Also, our algorithm terminates if the centre of min-disk is found during a call to the procedure CONSTRAINED-CENTRE in LOCALISE. So we only need to consider the case in which the algorithm does not terminate in this manner.

We first compute the weighted median slope, s_m, of the non-vertical lines in $\mathcal{L}$ and then divide the lines into three sets, $\mathcal{L}_>$, $\mathcal{L}_<$ and $\mathcal{L}_=$, having slope greater, smaller than and equal to s_m respectively. Let $\mathcal{L}_y$ be the set of vertical lines in $\mathcal{L}$. We pair the lines in $\mathcal{L}_>$ and $\mathcal{L}_<$ arbitrarily using their weights. We then compute intersection points of these pairs. The weights of both members of the pair should be equal and this weight is assigned to the intersection point. For example, a line with weight 3 can be paired twice — once with a line of weight 2 and second time with a line of weight 1 whereas the intersection points have weights 2 and 1 respectively. We can ensure the number of pairs to be linear as all the lines have integral weights only. Note that if the total weights of the sets $\mathcal{L}_>$ and $\mathcal{L}_<$ are unequal then some of the lines may remain unpaired.

Next we compute a vertical line, L_y, which divides the total weight of the intersection points of the above pairs and the vertical lines in $\mathcal{L}_y$ into two equal halves. We localise the centre of min-disk in one of the halfplanes determined by L_y, by using the solution of the constrained centre problem with L_y as the constraint line. Let J_y be the halfplane determined by L_y in which the centre of min-disk lies.

We then compute a line L_m with slope s_m which divides the total weight of points and lines — intersections points of the above pairs which lie outside J_y, and lines of $\mathcal{L}_=$ — into two equal halves. We again locate the constrained centre on this line and determine on which side of it the centre of min-disk lies. Let J_m be this halfplane so that the centre of min-disk now lies in $J_y \cap J_m$.

It is easy to see that at least one eighth of the total weight of lines in $\mathcal{L}$ are disjoint from $J_y \cap J_m$. These lines cross the opposite quadrant $\overline{J_y} \cap \overline{J_m}$.

The description of `LOCALISE` is as follows:

Algorithm 2 `LOCALISE`

```
Input:  Set of Lines $\mathcal{L}$
Output:  Localised Region $J$
begin
    Compute the intersections of an arbitrarily (equal weighing)
      paired set of lines in $\mathcal{L}$, one of larger slope
```

```
        than the weighted median slope and the other of smaller slope

      Localise Centre in $J_y$ with respect to a vertical line which
        halves the total weight of intersection points and
        vertical lines in $\mathcal{L}$

      Localise Centre in $J_m$ with respect to a line of weighted median
        slope which further halves the total weight of intersection
        points in $\overline{J_y}$ and weighted median slope lines in $\mathcal{L}$.

      $J \longleftarrow J_m \cap J_y$
      Return $J$
end ∎
```

Once we have localised the centre of min-disk, we repeatedly apply Theorem 3.1 and Theorem 3.4 on wedges, halfplanes, rays, lines and points to prune/replace the objects in the collection $\mathcal{C}$. We discuss separately the cases of points, lines, rays, halfplanes and wedges.

Let $\mathcal{P}$, $\mathcal{L}$, $\mathcal{R}$, $\mathcal{H}$ and $\mathcal{W}$ be the subsets of $\mathcal{C}$ containing all points, lines, rays and wedges in $\mathcal{C}$ respectively.

Points: We pair up the points in $\mathcal{P}$ arbitrarily and compute the perpendicular bisectors of these pairs. We use procedure LOCALISE with this set of bisectors and filter one point corresponding to every bisector that does not intersect the localised region. Per iteration this filters out at least $1/16$ of the points (Fig. 20(a): The centre is localised in *L*ower *L*eft quadrant, *LL* and the point-bisector crosses *U*pper *R*ight quadrant, *UR*).

Lines: Lines are filtered by localising the region containing the centre of min-disk in a quadrant defined by the bisector(s) of a pair of lines. When the lines are parallel the distance bisector is a line parallel to and equidistant from them.

We will call a pair of angular bisectors mates of each other in the following discussion. When we invoke procedure LOCALISE with this set of bisectors as input, there is no way to ensure that there exists at least a fraction of these bisector pairs which do not intersect LL. So, we have to do the localisation twice in the following way.

We first pair the set of lines in $\mathcal{L}$ arbitrarily and compute the bisector(s) of these pairs. We localise the region containing the centre of min-disk with respect to this set of bisectors. Let LL be this localised region. We consider all the bisectors that do not intersect LL. In the worst case, all these are angular bisectors such that their mates intersect LL. We invoke LOCALISE again with these intersecting mates as input. A fraction of these mate-bisectors do not intersect the localised region, say LL', returned by this second invocation of LOCALISE. These together with there mates from the previous invocation of LOCALISE do not intersect region $LL \cap LL'$ which contains the centre of min-disk. We filter one of the lines of each pair of lines whose distance bisector(s) does not intersect $LL \cap LL'$. This filters out $1/64$ of the lines in $\mathcal{L}$ (Fig. 20(b)).

Rays: Rays in $\mathcal{R}$ are replaced by localising the region containing the centre of min-disk in one of the halfplanes defined by the normal at the tail point of a ray. We do this by invoking LOCALISE with the set of normals at tail points of rays as input. This replaces at least $1/8$ of the rays in $\mathcal{R}$ (Fig. 20(c)).

Halfplanes: Halfplanes in $\mathcal{H}$ are filtered/replaced by their boundary lines by localising the region containing the centre of min-disk either in the interior of these or in the interior of their complement. We do this by invoking LOCALISE with the set of boundary lines. In each iteration at least $1/8$ of the halfplanes are filtered/replaced in this way.

Wedges: Wedges in $\mathcal{W}$ are filtered/replaced by localising the region containing the centre of min-disk in one of the four quadrants at the apex of a wedge (Fig. 20(d)). We then replace or filter the wedge, as the case might be. We call the four lines, two sides and two normals, associated with each wedge to be mates of one another.

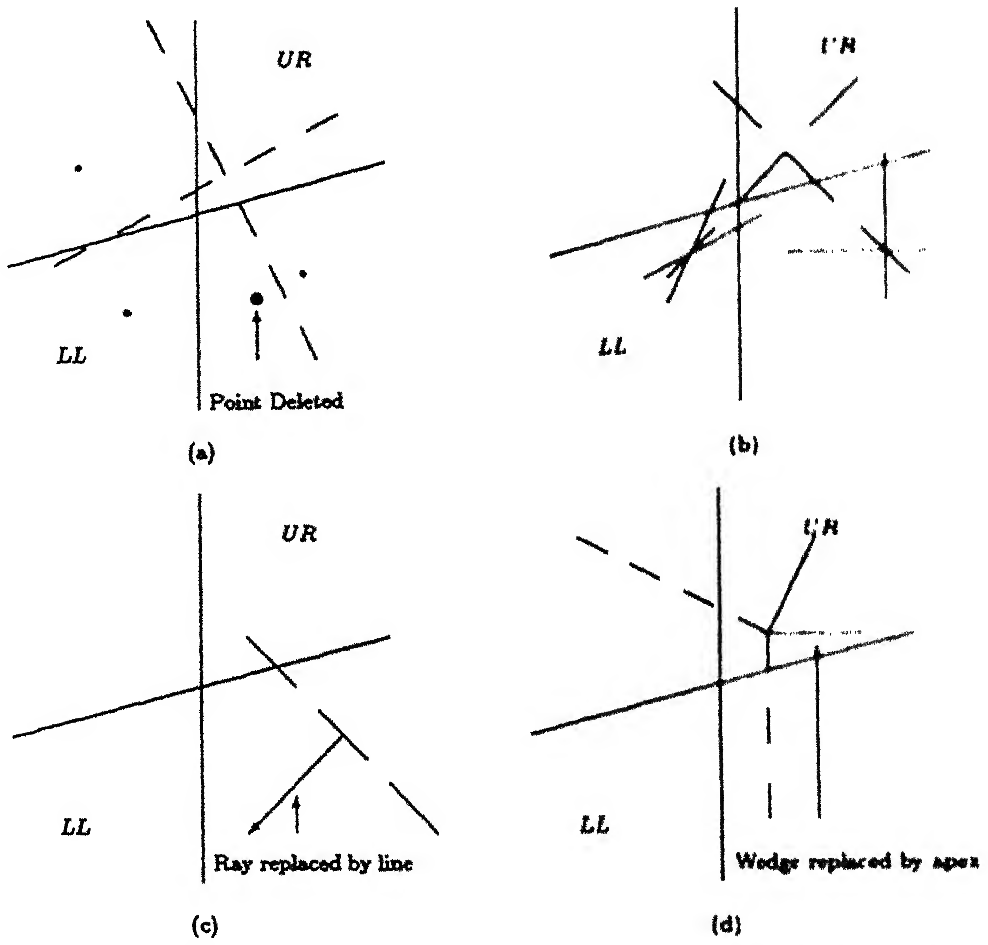

Figure 20: Localisation of unconstrained centre in LL with respect to derived lines

> We proceed exactly as in the case of line bisectors by first invoking LOCALISE with the set of derived lines of the wedges as input. Since for crossing derived line in the UR quadrant at most three derived lines of the corresponding wedge can intersect LL, we invoke LOCALISE twice. The first call is with the line/middle of these intersecting lines. Let LL' be the region returned. We invoke LOCALISE again with the remaining line, if any, that intersects LL' and whose mates do not. Let LL'' be the region that is output. Thus we have a fraction of wedges in $\mathcal{W}$ whose associated lines do not intersect $LL \cap LL' \cap LL''$. This replaces/filters at least $1/256$ of the wedges.

We can get an $O(N_C \log N_C)$ algorithm for computing the min-disk if we first convert all the wedges, halfplanes and rays to lines/points as above and then compute the min-disk of the resulting set consisting of lines and points only. To obtain a linear time algorithm, we do not differentiate between the bisectors of points and lines, associated lines of rays, halfplanes and wedges in the invocations of LOCALISE.

We classify the derived lines as active or inactive in a similar way as in the constrained case. We assign weights to the active derived line in exactly same way as the constrained case. At the beginning all the derived lines are active. We invoke LOCALISE with the set of active derived lines and if any active derived line does not intersect the localised region then we make it inactive for the rest of the computation. We replace/filter those objects in C none of whose derived lines are active.

We repeat these steps until no update occurs. We later show that this algorithm runs in linear time.

We give a description of the algorithm for the unconstrained centre problem below. We will denote the active derived lines by $\mathcal{D}$.

Algorithm 3 UNCONSTRAINED-CENTRE

Input: Bisectors of point-pairs and line-pairs, normals to rays, boundary lines of halfplanes, and normals to and sides of wedges

```
Output:  Min-disk
begin
    $\mathcal{D}$ ←—input set
    do
        LOCALISE with $\mathcal{D}$
        Update $\mathcal{D}$
        Filter/Replace objects in $C$
    while there is some update
    enddo
    Determine the centre by some brute force method
end  ∎
```

3.4 Analysis of the Algorithm

In this section we establish that the algorithm runs in linear time. First we wil analyse the constrained case and then the unconstrained one.

In every iteration of the algorithm some of the wedges, halfplanes, points and line are dropped or some of the wedges, halfplanes and rays are replaced by points o lines. The DAG above (Fig. 22) illustrates this conversion. Further each localisatio of J with respect to some derived line reduces the weight by at least one fourth For example, when a wedge having four intersections with J is converted to anothe wedge with three intersections with J then the net reduction in weight is 16 from 64 to 48. We show this reduction in Fig. 21. The number of intersections ar denoted in the subscripts, and the number of objects resulting from the conversio is written on top of arrows. Line pairs and point pairs are denoted by $\mathcal{LP}$ and $\mathcal{PP}$ respectively.

Let $N_{\mathcal{U}}$ denote the number of objects of type $\mathcal{U}$.

$$\begin{tikzcd}
 & & & & \mathcal{H}_1 \arrow[d] & \\
\mathcal{W}_4 \arrow[r, "1"] & \mathcal{W}_3 \arrow[r, "1"] & \mathcal{W}_2 \arrow[r, "1"] & \mathcal{W}_1 \arrow[r, "1/2"] \arrow[d, "1/2"'] & \mathcal{LP}_2 \arrow[r, "1"] & \mathcal{LP}_1 \arrow[l, bend left=60, "1/2"] \\
 & & & \mathcal{PP}_1 \arrow[loop below, "1/2"] & \mathcal{R}_1 \arrow[l, "1/2"'] \arrow[u, "1/2"] &
\end{tikzcd}$$

Figure 21: Reduction of Weights Per Iteration

$$\begin{tikzcd}
\emptyset & \emptyset & \\
\mathcal{P} \arrow[u] & \mathcal{W} \arrow[l] \arrow[u] \arrow[d] & \\
\mathcal{R} \arrow[u] \arrow[r] & \mathcal{L} \arrow[d] & \mathcal{H} \arrow[l] \arrow[d] \\
 & \emptyset & \emptyset
\end{tikzcd}$$

Figure 22: Conversion among Object Types

Theorem 3.5 *Algorithm* CONSTRAINED-CENTRE *is linear in* $N_\mathcal{P} + N_\mathcal{C} + N_\mathcal{R} + N_\mathcal{H} + N_\mathcal{W}$.

Proof:

Let $W_\mathcal{D}$ be the total weight of active derived lines.

At least one of the lines derived from an object or a combination of objects in C intersects L. Otherwise we can straightaway filter or replace the corresponding objects. Then we have following bounds on $W_\mathcal{D}$

$$\begin{aligned} W_\mathcal{D} &\geq 27N_\mathcal{W} + 27N_\mathcal{H} + 27N_\mathcal{R} + 27N_\mathcal{C}/2 + 18N_\mathcal{P}. \\ W_\mathcal{D} &\leq 48N_\mathcal{W} + 27N_\mathcal{H} + 27N_\mathcal{R} + 18N_\mathcal{C} + 18N_\mathcal{P}. \end{aligned}$$

Therefore,

$$W_\mathcal{D}/48 \leq \quad N_\mathcal{P} + N_\mathcal{C} + N_\mathcal{R} + N_\mathcal{H} + N_\mathcal{W} \quad \leq 2W_\mathcal{D}/27.$$

Hence

$$W_\mathcal{D} = \Theta\Big(N_\mathcal{P} + N_\mathcal{C} + N_\mathcal{R} + N_\mathcal{H} + N_\mathcal{W}\Big).$$

In every iteration the derived active lines of at least half the total weight are made inactive. The new derived lines of at most three fourth of this weight are added. So at least a fraction (around one eighth) of total weight is reduced. The pruning step is also linear in the number of objects which has the same complexity as the number of active derived lines. Therefore the algorithm is linear in $W_\mathcal{D}$. Hence the algorithm CONSTRAINED-CENTRE is linear in $N_\mathcal{P} + N_\mathcal{C} + N_\mathcal{R} + N_\mathcal{H} + N_\mathcal{W}$ as claimed.
∎

The procedure LOCALISE, which locates the unconstrained centre with respect to constant fraction of weighted lines, uses the algorithm CONSTRAINED-CENTRE as its basic routine. So, linearity of latter implies the linearity of former. We now prove a linear time bound for the algorithm UNCONSTRAINED-CENTRE.

Theorem 3.6 *Algorithm* UNCONSTRAINED-CENTRE *is linear in* $N_{\mathcal{W}} + N_{\mathcal{H}} + N_{\mathcal{R}} + N_{\mathcal{L}} + N_{\mathcal{P}}$.

Proof:

Let $W_{\mathcal{D}}$ be the number of active derived lines in the algorithm UNCONSTRAINED-CENTRE. Similar to the constrained case above, at least one of the derived lines of each object or the combination of objects in $\mathcal{C}$ is active. Otherwise we filter or replace the corresponding object/objects. Thus we have following bounds on $W_{\mathcal{D}}$

$$\begin{aligned} W_{\mathcal{D}} &\geq 27N_{\mathcal{W}} + 27N_{\mathcal{H}} + 27N_{\mathcal{R}} + 27N_{\mathcal{L}}/2 + 18N_{\mathcal{P}}, \\ W_{\mathcal{D}} &\leq 64N_{\mathcal{W}} + 27N_{\mathcal{H}} + 27N_{\mathcal{R}} + 18N_{\mathcal{L}} + 18N_{\mathcal{P}}/2. \end{aligned}$$

Therefore,

$$W_{\mathcal{D}}/64 \leq \quad N_{\mathcal{P}} + N_{\mathcal{L}} + N_{\mathcal{R}} + N_{\mathcal{H}} + N_{\mathcal{W}} \quad \leq 2W_{\mathcal{D}}/27,$$

and hence,

$$W_{\mathcal{D}} = \Theta\Big(N_{\mathcal{P}} + N_{\mathcal{L}} + N_{\mathcal{R}} + N_{\mathcal{H}} + N_{\mathcal{W}}\Big).$$

The pruning step is linear in the number of objects in $\mathcal{C}$. Furthermore, in every iteration, at least one eighth of the total weight of derived active lines is made inactive and new active lines of at most $^3/_4$ weights are added, with the consequence that the total weight is reduced by $^1/_{32}$. Therefore the algorithm is linear in $W_{\mathcal{D}}$.

Hence the algorithm CONSTRAINED-CENTRE is linear in $N_{\mathcal{W}} + N_{\mathcal{H}} + N_{\mathcal{R}} + N_{\mathcal{L}} + N_{\mathcal{P}}$ as claimed. ∎

3.5 Concluding Remarks

In this chapter, we have described an optimal algorithm for computing the smallest disk that intersects a finite collection of geometrical objects, containing convex polygonal disks, wedges, halfplanes, lines, points, line segments and rays by combining the prune and search technique of Megiddo with the novel idea of replacing complex geometrical objects by simpler ones

It would be worth investigating whether this approach can be used to compute the smallest intersecting disk of a collection of objects that includes simple polygons and also look at higher dimensional generalisations of the problems studied here

In the next chapter we again see one more application of the modified prune and search, which was used in this chapter, in the computation of a centre point for a finite planar set of points.

Chapter 4

Computing a Centrepoint of a Finite Planar Set of Points in Linear Time

The notion of a centrepoint of a finite set of points in two and higher dimensions is a generalisation of the concept of the median of a set of reals. In this chapter, we present a linear time algorithm for computing a centrepoint of a set of n points in the plane, which is optimal compared to the $O(n \log^3 n)$ complexity of the previously best known algorithm. We use suitable modifications of the ham-sandwich cut algorithm in [23] and the prune and search technique of Megiddo [21] to achieve this improvement.

4.1 Introduction

We all have an intuitive idea as to what phrases like "the very centre of the square" or "the very centre of the city" mean. To capture this intuition in a quantitative way, the *centre* of a set of n points, $\mathcal{P}$, in $\Re^d$ is defined as the maximal subset of $\Re^d$

such that any closed halfspace intersecting this subset contains at least $\lceil n/(d+1)\rceil$ points of $\mathcal{P}$ [35]. This subset is non-empty for any finite configuration of points (see, for example, [13]). Furthermore, it is closed and convex. A *centrepoint* is a member of the centre of $\mathcal{P}$.

On the real line $\Re$, a centrepoint is no other than a median of $\mathcal{P}$. Thus a centrepoint can be viewed as a generalisation of the median of a set of reals. On the other hand, the centre can also be viewed as a particular k-hull of $\mathcal{P}$. The *k-hull* of $\mathcal{P}$ is a maximal subset (closed and convex) of $\Re^d$ such that any closed half space intersecting this subset contains at least k points of $\mathcal{P}$. For instance, the 1-hull of $\mathcal{P}$ is its convex hull and the centre is its $\lceil n/(d+1)\rceil$-hull. The property of balanced partitioning makes the centre point useful for efficient divide and conquer algorithms in geometrical computing and large scale scientific computing[26, 32, 25, 36]. Recently Donoho and Gasko have suggested that centre point can be used as "robust" and high "breakdown point" estimators for multivariate datasets [10].

The interesting algorithmic problem of computing a centrepoint has been considered by various researchers. Cole et al gave an $O(n \log^5 n)$ algorithm for computing a centrepoint of a planar set of points [9]. Subsequently, Cole improved this bound to $O(n \log^3 n)$, using the powerful technique of slowing down a sorting network [8]. In this chapter, we propose an optimal linear time algorithm for computing a centrepoint of a planar set of points by using suitable modifications of the ham-sandwich cut algorithm for a pair of separable point sets [23] and the prune and search technique of Megiddo [21].

Linear time algorithms, however, were known for computing an *approximate* or *ε-centrepoint* [19, 32, 23]. We obtain this weaker type of centrepoint if we decrease the lower bound, in the above definition of the centre, to $\lceil n(1-\varepsilon)/(d+1)\rceil$, where $0 < \varepsilon < 1$. Actually, Megiddo [23] only gave an algorithm for computing a partition of a (planar) set of n points with two lines such that each closed quadrant contains at least $\lfloor n/4 \rfloor$ points. An algorithm for computing an ε-centrepoint, where $0 < \varepsilon \leq 1/4$, is implicit in this.

The thesis proposes an optimal algorithm for computing a centrepoint of a planar set of points by using an interesting modification of Megiddo's prune and search technique[21]. This consists of adding a few extra points in each pruning step so that a subspace of the original solution space is retained, while ensuring a net deletion of points. In the description of our algorithm, we assume the usual RAM model of computation; the point set $\mathcal{P}$, however, is not assumed to be in general position.

This chapter is organised as follows. In section 4.2, we discuss which points to prune. In section 4.3, we describe the method used to find these points. The algorithm is presented in section 4.4. Section 4.5 contains an analysis of the time complexity of the algorithm. Concluding remarks are given in section 4.6.

4.2 What to Prune

Let $\mathcal{P}$ be a finite set of points in the plane. In the subsequent discussion we use the following notations. We denote the centre of $\mathcal{P}$ by $\textsc{centre}(\mathcal{P})$ and the k-hull of $\mathcal{P}$ by $\textsc{hull}(k, \mathcal{P})$. We use the notations $\mathcal{P}_H$, $\mathcal{P}_{GH}$, $\mathcal{P}_{FGH}$, ... to denote the points of $\mathcal{P}$ contained in H, $G \cap H$, $F \cap G \cap H$,... respectively, where F, G, H, ... are any closed or open halfplanes. We denote the complement of a set S by $\overline{S}$. As we frequently need to use the numbers $\lceil|\mathcal{P}|/3\rceil$ and $\lceil|\mathcal{P}|/3\rceil - \lceil|\mathcal{P}|/4\rceil$ in this and the following sections, we denote these by $N_{\mathcal{P}}$ and $M_{\mathcal{P}}$ respectively.

The basic idea of our algorithm is to use the prune and search strategy of Megiddo [21]. Clearly, we cannot hope to compute $\textsc{centre}(\mathcal{P})$ by a naive application of this technique, since the centre of a reduced set need not be the same as the centre of the original set. However, it might be possible to prune points in such a way that the centre of the pruned set is a subset of the centre of the original set. If so, by repeated pruning we may at least be able to compute a centrepoint, if not some larger subset. Below we show that this is indeed possible, and as a first step towards this goal we make the following important observation.

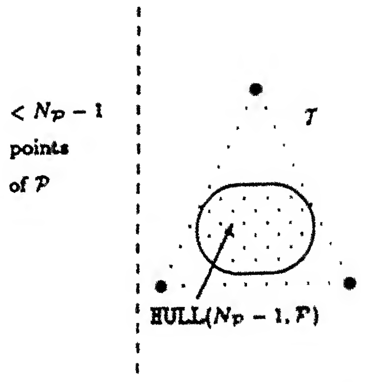

Figure 23: CENTRE($\mathcal{P} - \mathcal{T}$) $\subseteq$ CENTRE($\mathcal{P}$)

Observation 4.1 *If $\mathcal{T}$ is the set of vertices of a triangle that contains* CENTRE($\mathcal{P}$), *then* CENTRE($\mathcal{P}$) *is a subset of* CENTRE($\mathcal{T} \cup \mathcal{P}$).

Proof: Let c be any centrepoint of $\mathcal{P}$, i.e. $c \in$ CENTRE($\mathcal{P}$). By definition, any closed halfplane, say H, that contains c also contains at least $N_{\mathcal{P}}$ points of $\mathcal{P}$. Since by assumption c is contained in the triangle formed by $\mathcal{T}$, we have $\mathcal{T} \cap H \neq \emptyset$ so that H contains at least one point of $\mathcal{T}$. Thus H contains at least $N_{\mathcal{P}} + 1 = N_{\mathcal{P} \cup \mathcal{T}}$ points of $\mathcal{P} \cup \mathcal{T}$. Since H is arbitrary, it follows that c is in CENTRE($\mathcal{P} \cup \mathcal{T}$). ∎

The above observation has the following important consequence. If we can find a set of three points, $\mathcal{T}$, in $\mathcal{P}$ such that the triangle formed by these points contains the centre of $\mathcal{P} - \mathcal{T}$, then by discarding these three points we can achieve the goal of ensuring that the centre of the pruned set is a subset of the centre of the original set. The following lemma gives a sufficient characterisation of such a triplet of points.

Lemma 4.1 *Let $\mathcal{T}$ be three points of $\mathcal{P}$ such that* HULL($N_{\mathcal{P}} - 1, \mathcal{P}$) *is contained in the (closed) triangle formed by $\mathcal{T}$. Then* CENTRE($\mathcal{P} - \mathcal{T}$) *is a subset of* CENTRE($\mathcal{P}$).

Proof: Let c be a centrepoint of $\mathcal{P} - \mathcal{T}$ and T be the triangle formed by $\mathcal{T}$. We claim that c lies inside T. Otherwise, if c lies outside T, and therefore outside HULL($N_{\mathcal{P}} - 1, \mathcal{P}$), then there exists an open halfplane that contains c and at the same

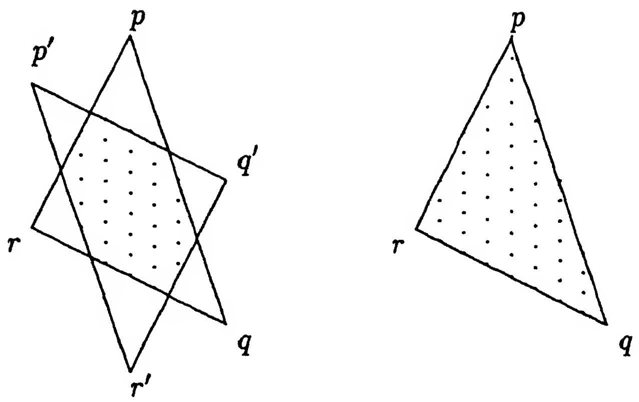

Figure 24: Removal of three points may expand the centre

time contains less than $N_{\mathcal{P}} - 1$ points of $\mathcal{P}$. It can be easily seen that this halfplane contains less than $N_{\mathcal{P}} - 1 = N_{\mathcal{P}-\mathcal{T}}$ points of $\mathcal{P} - \mathcal{T}$ (Fig. 23). This contradicts the assumption that c is a centrepoint of $\mathcal{P} - \mathcal{T}$. Hence c is contained in T and therefore CENTRE($\mathcal{P} - \mathcal{T}$) is also contained in T.

The proof of the result now follows from Observation 4.1. ∎

Remark. We would like to point out a subtlety involved here. Had we chosen the triangle $\mathcal{T}$ to contain CENTRE($\mathcal{P}$) instead, we could not have guaranteed the conclusion of the above lemma. Fig. 24 shows why.

The above lemma suggests an algorithmic solution to the problem of computing a triplet of points that can be pruned. Since an open halfplane that contains less than k points of $\mathcal{P}$ does not interesct HULL($k, \mathcal{P}$), we find three open halfplanes, each containing less than $N_{\mathcal{P}} - 1$ points of $\mathcal{P}$ and situated so that the intersection of their complements is a bounded triangle. This triangle contains HULL($N_{\mathcal{P}} - 1, \mathcal{P}$). If this triangle is of non-zero area then a required triplet is formed by choosing a point each from the closure of the pairwise interesctions of these halfplanes (Fig. 25).

The snag in this solution is that there are configurations of points for which we cannot find such a triplet for any choice of these open halfplanes. An example of

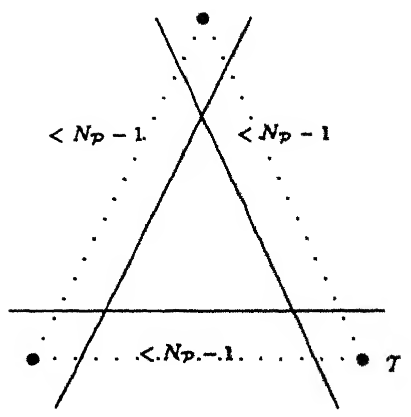

Figure 25: Pruning of triplets $\mathcal{T}$ from $\mathcal{P}$

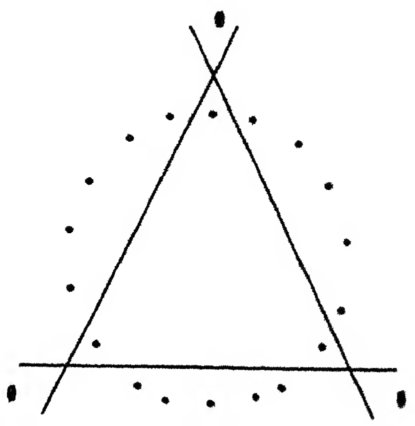

Figure 26: A pathological configuration

such a configuration is shown in Fig. 26, where the points are evenly arranged on the circumference of a circle.

To overcome this problem, we enlarge the scope of the above lemma, allowing for the choice of four points. For this we briefly review the concept of a *Radon point*. Any set of at least four points in the plane can be partitioned into two disjoint subsets such that the intersection of their convex hulls is non-empty. A *Radon point* of this set is a point in this intersection. A Radon point of four points is unique when these points are vertices of a quadrilateral of non-zero area (Fig 27).

Lemma 4.2 *Let $\mathcal{Q}$ be any four points of $\mathcal{P}$ such that the (closed) convex hull of $\mathcal{Q}$*

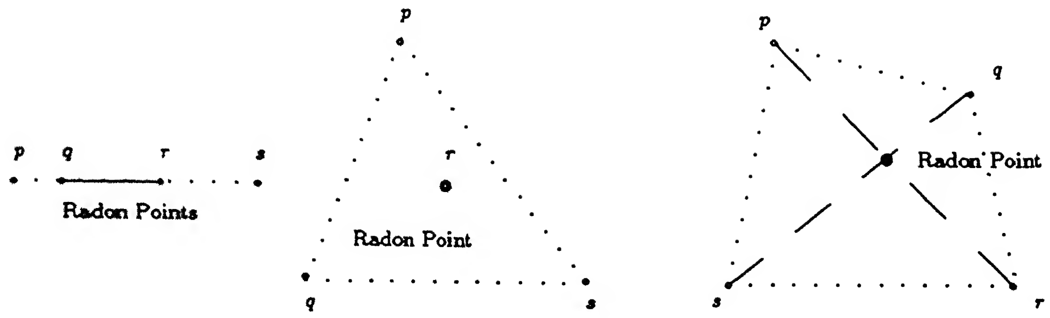

Figure 27: Radon point(s) of four points p, q, r and s

contains $\text{HULL}(N_{\mathcal{P}} - 1, \mathcal{P})$. *Then* $\text{CENTRE}((\mathcal{P} - \mathcal{Q}) \cup \{q\})$ *is a subset of* $\text{CENTRE}(\mathcal{P})$, *where q is the Radon point of* $\mathcal{Q}$.

Proof: Let c be a centrepoint of $(\mathcal{P} - \mathcal{Q}) \cup \{q\}$. Consider any closed halfplane H that contains c. Then, by definition, it contains at least $N_{(\mathcal{P}-\mathcal{Q})\cup\{q\}}$ points of $(\mathcal{P} - \mathcal{Q}) \cup \{q\}$.

We claim that c lies in the convex hull, Q, of $\mathcal{Q}$. Let p be a point that lies outside Q, and therefore outside $\text{HULL}(N_{\mathcal{P}} - 1, \mathcal{P})$. It is then possible to find an open halfplane that contains p and contains less than $N_{\mathcal{P}} - 1 = N_{(\mathcal{P}-\mathcal{Q})\cup\{q\}}$ points of $(\mathcal{P} - \mathcal{Q}) \cup \{q\}$. Hence p is not a centrepoint of $(\mathcal{P} - \mathcal{Q}) \cup \{q\}$. Therefore c cannot lie outside Q.

To complete the proof, we have to show that any closed halfplane H which contains c contains at least $N_{\mathcal{P}}$ points of $\mathcal{P}$. Clearly, H contains at least $N_{(\mathcal{P}-\mathcal{Q})\cup\{q\}}$ points of $(\mathcal{P} - \mathcal{Q}) \cup \{q\}$. Three different cases arise, depending on the relative positions of the points of $\mathcal{Q}$.

Case 1: The four points in $\mathcal{Q}$ form a non-convex quadrilateral.

This case is a trivial application of Lemma 4.1. The three convex vertices of Q form a triangle that encloses $\text{HULL}(N_{\mathcal{P}} - 1, \mathcal{P})$ and the concave vertex is q. Thus by Lemma 4.1 $\text{CENTRE}((\mathcal{P} - \mathcal{Q}) \cup \{q\}) \subseteq \text{CENTRE}(\mathcal{P})$.

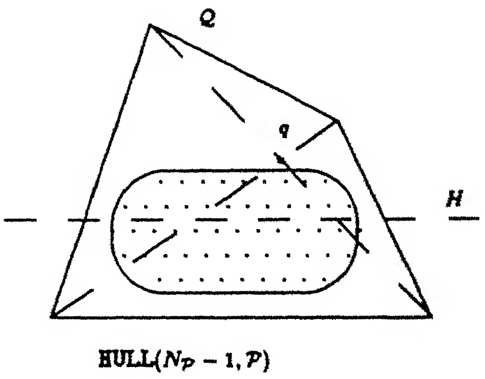

Figure 28: Substitution of Q by its Radon point q

Case 2: The four points of Q form a convex quadrilateral but their Radon point q does not belong to H.

Since the quadrilateral Q and the halfplane H both contain c, their intersection is non-empty. Thus H contains at least one of the vertices of Q and therefore at least $N_{\mathcal{P}} = N_{(\mathcal{P}-Q)\cup\{q\}} + 1$ points of $\mathcal{P}$.

Case 3: The four points of Q form a convex quadrilateral and their Radon point q belongs to H (Fig. 28).

In this case H contains at least two points of Q. We can therefore delete q from H and still claim that H contains at least $N_{\mathcal{P}}$ points of $\mathcal{P}$. ∎

Thus in all cases H contains at least $N_{\mathcal{P}}$ points of $\mathcal{P}$. Since H is arbitrary, c is a centrepoint of $\mathcal{P}$ as well. Hence CENTRE$((\mathcal{P} - Q) \cup \{q\})$ is a subset of CENTRE$(\mathcal{P})$.
∎

The above lemma is the cornerstone of our pruning mechanism. In the next section we will show how to use ham-sandwich cuts to make a clever choice of four open halfplanes so that we can prune a fraction of the input set by repeatedly applying the last two lemmas.

4.3 How to prune

In this section and afterwards, we use the words left, right, up and down, wherever these are unambiguous, to simplify the arguments.

Suppose we choose four open halfplanes, call them L, U, R and D, (mnemonics for *Left, Up, Right* and *Down* respectively) such that each contains less that $N_{\mathcal{P}} - 1$ points of $\mathcal{P}$ and its closure at least $N_{\mathcal{P}}$ points, and together they enclose a non-zero bounded area. Why do we expect this choice to give us a triplet/quadruple of points $\mathcal{Q}$ satisfying the conditions of Lemma 4.1/4.2 ? We give an intuitive justification of this below.

If the pairwise intersctions of "adjacent" halfplanes (i.e. L and U, U and R, etc.) are empty we would get a configuration as shown in Fig. 29. In this configuration, the total number of points in all the halfplanes taken together exceeds the total number of points in $\mathcal{P}$ by approximately one-third! This is impossible. So we might attempt to construct the four halfplanes in such a way that this excess is distributed evenly among the pairwise intersections of the adjacent halfplanes and thereby obtain approximately $M_{\mathcal{P}}$ triplets/quadruples of points satisfying the conditions of Lemma 4.1/4.2.

It is possible to do this as the construction below shows.

4.3.1 Computation of Open Halfplane L

We fix L as follows. We determine an extreme point p of $\mathcal{P}$ with minimum abscissa, and join all the remaining points to it. We compute the line that passes through p such that its slope is the $N_{\mathcal{P}} - 1$-th largest of the slopes of the above lines. The open halfplane above this line is chosen to be L. Clearly, it takes linear time to compute L. This way we make sure that L contains less than $N_{\mathcal{P}} - 1$ points and its closure contains at least these many points of $\mathcal{P}$. Moreover, this halfplane contains

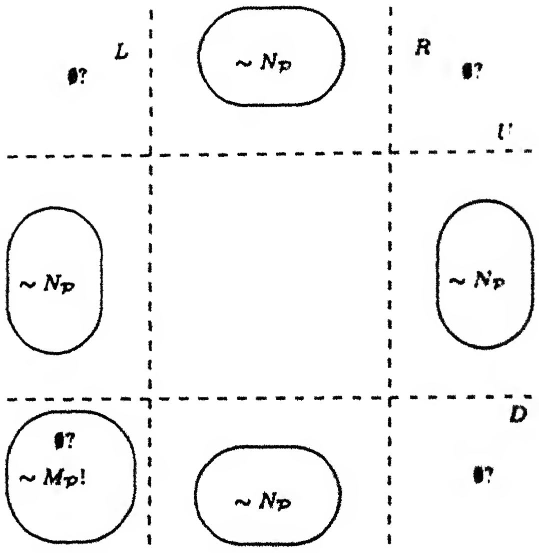

Figure 29: The intuition behind pruning

at least two points on its boundary. Really speaking, this latter requirement is not necessary but it helps us to treat the four halfplanes uniformly in the analysis of the algorithm.

4.3.2 Computation of Open Halfplane U

Since the point set $\mathcal{P}$ can be degenerate we need to be careful in the construction of U in order that none of the closed quadrants determined by the boundaries of L and U contains too few points of $\mathcal{P}$. This will also result in an even distribution of points among the pairwise intersections of adjacent halfplanes.

To achieve this we use the ham-sandwich cut algorithm of Megiddo [23]. However, the ratios in which we propose to divide the point sets are arbitrary. As we show below, Megiddo's algorithm can be easily adapted to take care of this aspect.

As is usual, we consider the dual problem, letting the bondary of L be the y-axis in the primal plane. Under the duality transformation that we consider, points that lie on the boundary of L map to horizontal lines; the set of points that lie in L

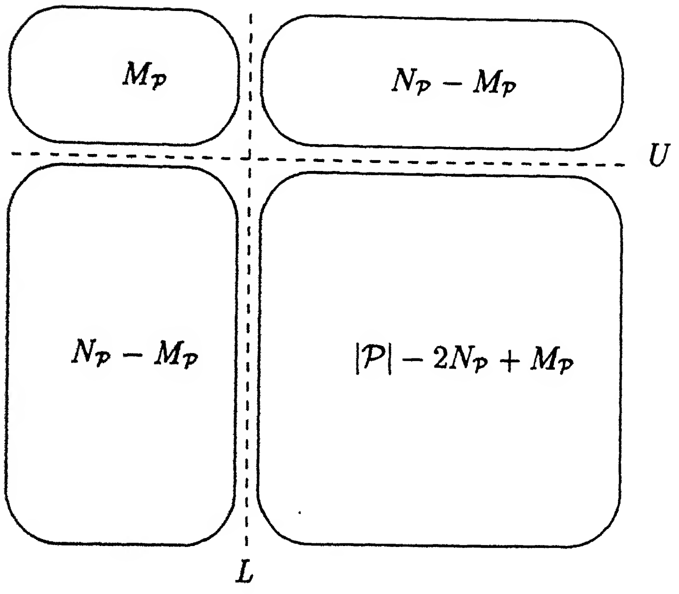
$M_{\mathcal{P}}$
$N_{\mathcal{P}} - M_{\mathcal{P}}$
U
$N_{\mathcal{P}} - M_{\mathcal{P}}$
$|\mathcal{P}| - 2N_{\mathcal{P}} + M_{\mathcal{P}}$
L

Figure 30: Computation of L and U

map to a set of lines, $\mathcal{L}_-$, with negative slopes; and the set of points that lie in interior of $\overline{L}$ map to a set of lines, $\mathcal{L}_+$ with positive slopes. In order to get the type of ham-sandwich cut we want, we put the horizontal lines in $\mathcal{L}_-$. Let us assume that $\mathcal{L}_+$ contains at least $N_{\mathcal{P}} - M_{\mathcal{P}}$ lines in the dual plane. It can be seen that the boundary of U corresponds to that point in the dual plane which has less than $M_{\mathcal{P}}$ lines of $\mathcal{L}^-$ above it; at least these many lines of $\mathcal{L}^-$ passing through or above it; less than $N_{\mathcal{P}} - M_{\mathcal{P}}$ lines of $\mathcal{L}^+$ above it; and at least these many lines of $\mathcal{L}^+$ passing through or above it. We compute this point as follows.

It can be easily seen that we can resolve a query for any positive (negative) slope query line in Megiddo's method as follows. We first compute the $M_{\mathcal{P}}$-th ($N_{\mathcal{P}} - M_{\mathcal{P}}$-th) intersection of the lines in $\mathcal{L}^-$ ($\mathcal{L}^+$) with the query line. We then count the number of lines in $\mathcal{L}^+$ ($\mathcal{L}^-$) lying strictly above and the number of lines in $\mathcal{L}^+$ ($\mathcal{L}^-$) passing through this intersection point. If the sum of these two counts is smaller than $N_{\mathcal{P}} - M_{\mathcal{P}}$ ($M_{\mathcal{P}}$) then the solution point lies below the query line. If the first count is greater than or equal to $N_{\mathcal{P}} - M_{\mathcal{P}}$ ($M_{\mathcal{P}}$) then it is above the query line, else it is same as the intersection point. By thus changing the method of query resolution, we get the solution point in linear time. This gives us the required halfplane U in the primal plane (Fig. 30).

4.3.3 Computation of Open Halfplanes D and R

The open halfplane D is determined with respect to L in a similar manner by changing "above" to "below" throughout the above discussion. Thus we make sure that D contains the "lower end" of the boundary of L whereas U contains the "upper end".

The halfplane R is also determined similarly except that U plays the role of L here. We also ensure that R contains the "right end" of the boundary of U whereas L contains the "left end".

The idea behind choosing D and R in this way is to make the boundaries of the halfplanes U, R, D and L the adjacent sides of a bounded quadrilateral such that the halfplanes face "outwards". This is so because if the interior of $\mathcal{I}$ contains at least $N_{\mathcal{P}} - M_{\mathcal{P}}$ points of $\mathcal{P}$ then the boundaries of both U and D are disjoint from the boundary of L and it can be seen that the intersection of the complements of the above halfplanes is bounded. As explained before, it can also be seen that $R \cap D$ contains non-zero points of $\mathcal{P}$. Moreover, since each of these half planes contain less than $N_{\mathcal{P}} - 1$ points of $\mathcal{P}$, the intersection of the complements of these encloses HULL$(N_{\mathcal{P}} - 1, \mathcal{P})$.

However, in the computation of U it is quite possible that the set $\mathcal{L}_+$ contains less than $N_{\mathcal{P}} - M_{\mathcal{P}}$ lines. The consequence of this is that the computed U may have the same boundary as that of L and thus the intersection of the complements of the open half planes may be unbounded. This is inadmissible in our algorithm. So we need to take care of this degenerate case separately.

4.3.4 The Degenerate Case

If there are more than $N_{\mathcal{P}}$ points on the boundary of L then the boundaries of U and D, as computed above, is the same as that of L. This degeneracy is detected in the algorithm when $\mathcal{L}_+$ contains less than $N_{\mathcal{P}} - M_{\mathcal{P}}$ lines i.e. interior of $\overline{L}$ contains less that $N_{\mathcal{P}} - M_{\mathcal{P}}$ points of $\mathcal{P}$. There is no loss of generality if we assume that L also contains less than $N_{\mathcal{P}} - M_{\mathcal{P}}$. Otherwise, we can switch the sides of L and let the interior of $\overline{L}$ be our new L. We will then be able to compute the other halfplanes, as required, with respect to this open half plane.

Let the interior of $\overline{L}$ be the open halfplane R. We compute the open halfplanes U and D as follows. We first distribute the alternate points of $\mathcal{P}$ on the boundary of L between the sets R and L. Let these resulting sets be $\mathcal{S}_R$ and $\mathcal{S}_L$ respectively. We then compute U and D by the ham-sandwich cut algorithm such that these satisfy the following properties: these contain less than total $N_{\mathcal{P}} - 1$ points of $\mathcal{P}$ and contain an equal number of points of $\mathcal{S}_R$ and $\mathcal{S}_L$. The halfplanes U and D are computed such that these contain the "opposite ends" of the boundary of L. It can be seen that if L, R, U and D are computed in this manner then each of the pairwise intersections of the adjacent halfplanes contains at least $M_{\mathcal{P}}$ (approximately $2M_{\mathcal{P}}$) points of $\mathcal{P}$. Since the intersection of the complements of these contains HULL($N_{\mathcal{P}} - M_{\mathcal{P}}, \mathcal{P}$), we can apply pruning at least $M_{\mathcal{P}}$ times similarly as in the non-degenerate case that we describe below.

The degeneracy of open halfplanes taken care of, we may assume safely that the open halfplanes L, U, R and D can be computed such that these meet our requirements.

4.3.5 The Pruning Step

We have been able to ensure by the construction of the halfplanes as above that the closure of each of the sets $L \cap U$, $L \cap D$ and $R \cap U$ contains at least $M_{\mathcal{P}}$ points of

$\mathcal{P}$. We shall prove later that the closure of $R \cap D$ also contains at least these many points. However, for the rest of this section we will assume this.

It is now clear how we can prune points. Two points of detail, however, must be noted. First, in order to ensure that the conditions of Lemmas 4.1 and 4.2 remain valid throughout the pruning step, we must choose a triple or a quadruple of points in such a way that, whenever there is a possibility that the conditions of the above Lemmas are violated in the successive pruning steps, we delete an interior point in an open halfplane. Second, to maximise the number of points that are pruned, we must ensure that no two points of a quadruple, selected for pruning, belong to either $R \cap L$ or $U \cap D$.

To implement the above observations we maintain the points that are candidates for pruning in six disjoint sets, viz., $\mathcal{P}_{LU}$, $\mathcal{P}_{UR}$, $\mathcal{P}_{RD}$, $\mathcal{P}_{LD}$, $\mathcal{P}_{LR}$ and $\mathcal{P}_{UD}$. The points on the boundaries are put in the relevant sets. So, the four sets, $L \cap U$, $L \cap D$, $R \cap U$ and $R \cap D$, are now effectively divided into six sets, three of which correspond to choices of triangles $\mathcal{T}$.

We discard the triangles $\mathcal{T}$ and substitute quadruples $\mathcal{Q}$ by their Radon points such that a maximum number of the above half planes contain an interior point. Substitution of $\mathcal{Q}$ is done as follows. If $\mathcal{Q}$ form a convex quadrilateral we delete it from $\mathcal{P}$ and add the intersection point of the diagonals to $\mathcal{P}$. Otherwise we delete the convex vertices but retain the concave one. We can repeat this pruning procedure on the reduced set of points thus obtained, since the halfplanes L, U, D and R continue to contain less than $N_{\mathcal{P}} - 1$ points of the reduced set $\mathcal{P}$, until one of the four sets is empty. We note that this reduces the size of $\mathcal{P}$ by approximately one fourth.

4.4 The Centrepoint Algorithm

It is now clear from the discussion in the previous sections, how we can find a centrepoint of $\mathcal{P}$.

In each iteration we compute the points that are to be discarded or replaced. By throwing away these points we reduce the size of the set by a non-zero fraction. When the size of the set becomes so small (of size at most 10) that no more points can be discarded we halt the pruning procedure and compute a centrepoint by any straight forward method.

The algorithm is given below.

Algorithm 4 CENTREPOINT

```
Input: Set of Points $\mathcal{P}$
begin
    do
        Compute the open halfplanes $L$, $U$, $D$ and $R$
        Update $\mathcal{P}$ by deleting $\mathcal{T}$ and replacing $\mathcal{Q}$ by
          their respective Radon points
    while there is some replacement/deletion
    enddo
    Compute a centrepoint by any bruteforce method
end ∎
```

We justify that anytime during the pruning step each halfplane contains less than $N_{\mathcal{R}} - 1$ points where $\mathcal{R}$ is the current set of points.

We first argue for a deleted point that is also an interior point of a halfplane. We first consider the case when four points are pruned and their Radon point is added to the set $\mathcal{P}$. If a point that is pruned lies in the interior of a halfplane and the Radon point does not lie in that halfplane then the number of points in this half plane is decreased by one. Since the total number of points decreases by three, the number of points is less than $N_{\mathcal{R}} - 1 = N_{\mathcal{P}} - 2$ where $\mathcal{R}$ is the new set. Now suppose that the Radon point also lies in this halfplane. Since a halfplane that contains a Radon point of four points contains at least two of these points, in this case also the

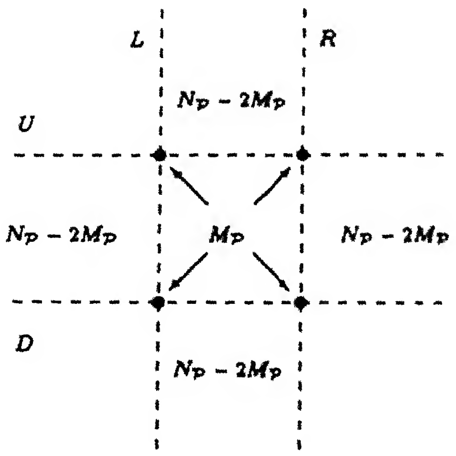
L
R
U
D
$N_\mathcal{P} - 2M_\mathcal{P}$
$N_\mathcal{P} - 2M_\mathcal{P}$
$M_\mathcal{P}$
$N_\mathcal{P} - 2M_\mathcal{P}$
$N_\mathcal{P} - 2M_\mathcal{P}$

Figure 31: The worst case of pruning!

number of points in the halfplane decreases by one. The same argument holds for triplets of points.

Now we consider the case when there is no interior point among the four pruned points in a halfplane. In this case the Radon point also does not belong to it. We consider the worst case in which for every choice of a quadruple or a triplet of points those chosen from this halfplane lie on its boundary. This case is simpler to analyse and it is easy to see that same argument extends to the other cases. It is clear that when we start pruning, the number of points in each of the open halfplanes is less than $N_\mathcal{P} - 1$ points. Let us see what happens when we have to prune the last triplet or quadruple of points after applying pruning $M_\mathcal{P} - 1$ times. The maximum number of points in this open halfplane is less than

$$N_\mathcal{P} - M_\mathcal{P}$$

since $M_\mathcal{P} - 1$ points in the corner quadrants are already pruned.

The total number of points at this instant is

$$|\mathcal{P}| - 3M_\mathcal{P} + 3$$

The number in the previous expression is exactly one less than the ceiling of this number. Thus the last set of points can also be pruned. It can easily be seen that in the intermediate steps also we can do the pruning.

The proof of correctness of this algorithm now follows from Lemmas 4.1 and 4.2. In the next section we give an analysis of the running time of this algorithm.

4.5 Analysis of the Centrepoint Algorithm

For the purpose of the proofs in this section, we assume that the boundaries of the halfplanes L, U, D and R do not contain any point of $\mathcal{P}$. This assumption is not necessary to establish the linearity of our algorithm but it simplifies the proof to a great extent. This can be achieved by slightly perturbing the points of $\mathcal{P}$ such that: no point migrates across the boundary of any halfplane; each corner region, such as $L \cap U$ etc., contains exactly $M_{\mathcal{P}}$ points of $\mathcal{P}$; and each halfplane contains at least $N_{\mathcal{P}}$ points of $\mathcal{P}$. Intuitively, such a perturbation does not matter because if we can prove that the perturbed set $\mathcal{P}_{RD}$ contains $M_{\mathcal{P}}$ points then these many points also belong to the closure of $\mathcal{P}_{RD}$ in the non-perturbed set $\mathcal{P}$. As a consequence we do not need to state explicitly whether the halfplanes are open or closed in the following discussion.

To prove that the algorithm is linear we have to show that the size of $\mathcal{P}$ is reduced by at least a fraction in each pruning step. We know by construction that each of the sets, $\mathcal{P}_{LU}$, $\mathcal{P}_{UR}$ and $\mathcal{P}_{LD}$, contains $M_{\mathcal{P}}$ points. We have to show that $\mathcal{P}_{RD}$ also contains at least $M_{\mathcal{P}}$ points in order to prune at least these many triangles/quadruples from $\mathcal{P}$. For this we will have to consider all the possible relative positions of the halfplanes L, U, R and D. Several cases arise. A few of these can be straight away discarded by using the following lemmas.

Lemma 4.3 *Let F, G and H be three halfplanes. Then $\overline{F} \cap \overline{G} \cap \overline{H}$ is a bounded triangle if and only if $F \cap G \subset \overline{H}$.*

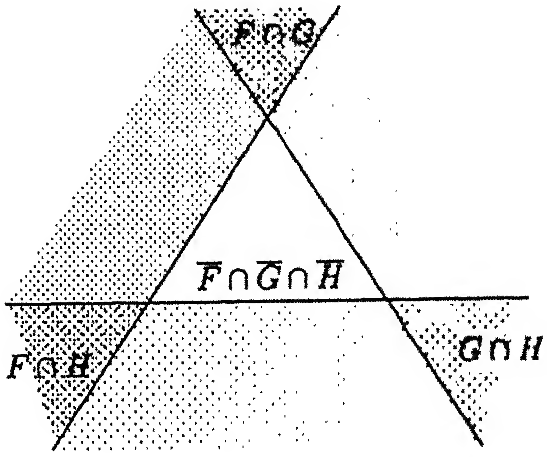

Figure 32: Intersection of three halfplanes

Proof: Straight forward(Fig. 32). ∎

Corollary 4.1 *$\overline{F} \cap \overline{G} \cap H$ is a bounded triangle if and only if $F \cap G \subset H$ if and only if $F \cap \overline{H} \subset \overline{G}$.*

The sets U, L, R and D satisfy some additional constraints also on account of their specific methods of construction.

Let us denote by p_{GH} the intersection point of the boundaries of any two halfplanes G and H. For the sake of simplicity, we may assume without any loss of generality that no three boundaries of the above halfplanes intersect at a point. If they do, then we can treat the said configuration in one of the cases discussed later on.

Lemma 4.4 *The intersection of the halfplanes U and D is contained in L if and only if p_{UD} is contained in L, i.e. $U \cap D \subset L \Longleftrightarrow p_{UD} \in L$.*

Similarly, $L \cap R \subset U \Longleftrightarrow p_{LR} \in U$.

Proof: We prove only the first equivalence. The proof of the second is similar.

($\Longrightarrow$) Easy.

($\Longleftarrow$) Let p_{UD} be in L. Then four cases arise, depending on the orientation of the halfplanes U and D.

Case 1: $U \cap \overline{D} \subset L$

Case 2: $\overline{U} \cap D \subset L$

These cases are not possible. For by construction, the halfplanes U and D contain the "opposite ends" of the boundary of L.

Case 3: $\overline{U} \cap \overline{D} \subset L$

This case is also impossible since it implies that the centre is empty.

Case 4: $U \cap D \subset L$

This is the only permissible case(Fig. 33). ∎

Hence proved. ∎

We have the following similar lemma when p_{UD} lies in $\overline{L}$.

Lemma 4.5 *The intersection of the halfplanes U and D is contained in $\overline{L}$ if and only if p_{UD} is contained in $\overline{L}$, i.e. $U \cap D \subset \overline{L} \Longleftrightarrow p_{UD} \in \overline{L}$.*

Similarly, $L \cap R \subset \overline{U} \Longleftrightarrow p_{LR} \in \overline{U}$.

Proof: Here too we prove only the first equivalence.

($\Longrightarrow$) Easy.

($\Longleftarrow$) Let p_{UD} be in $\overline{L}$. Then four different cases arise.

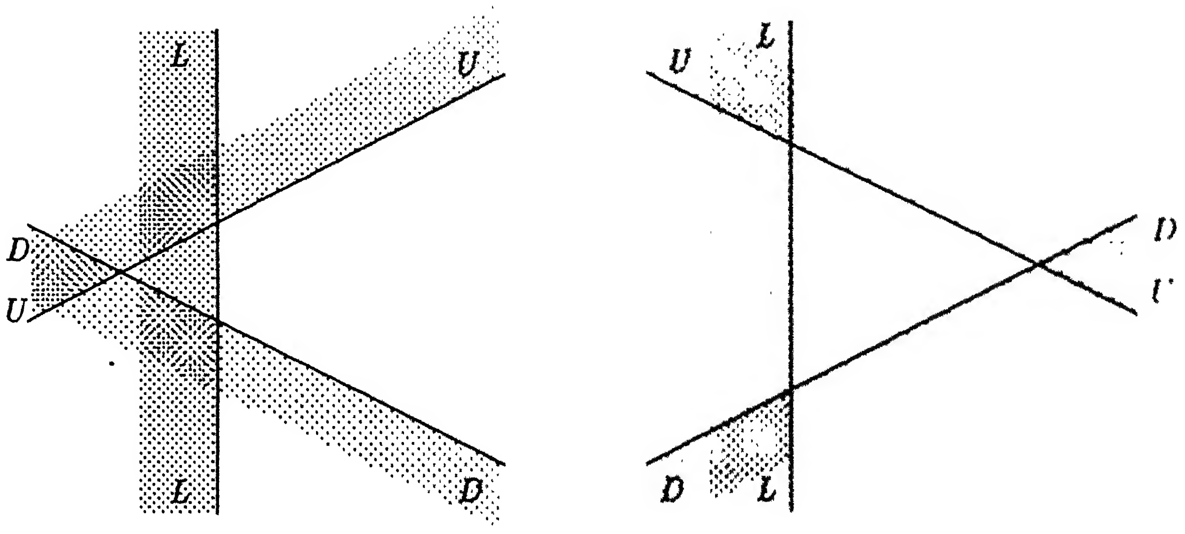

Figure 33: Either $U \cap D \subset L$ or $U \cap D \subset \overline{L}$

Case 1: $U \cap \overline{D} \subset \overline{L}$

Case 2: $\overline{U} \cap D \subset \overline{L}$

These cases are not allowed by construction.

Case 3: $\overline{U} \cap \overline{D} \subset \overline{L}$

To see that this case is also not permissible, let us count the number of points in $\mathcal{P}_L$. First we prove that $\mathcal{P}_L = \mathcal{P}_{LU} \cup \mathcal{P}_{LD}$.

$$\begin{aligned} \mathcal{P}_L &= \mathcal{P}_{LUD} \cup \mathcal{P}_{LU\overline{D}} \cup \mathcal{P}_{L\overline{U}D} \cup \mathcal{P}_{L\overline{U}\,\overline{D}} \\ &= \mathcal{P}_{LUD} \cup \mathcal{P}_{LU\overline{D}} \cup \mathcal{P}_{L\overline{U}D}, \qquad \text{since } \mathcal{P}_{L\overline{U}\,\overline{D}} = \emptyset, \\ &= \mathcal{P}_{LU} \cup \mathcal{P}_{LD}, \end{aligned}$$

since $\mathcal{P}_{LUD} \cup \mathcal{P}_{L\overline{U}D} = \mathcal{P}_{LD}$ and $\mathcal{P}_{LUD} \cup \mathcal{P}_{LU\overline{D}} = \mathcal{P}_{LU}$. Hence,

$$\begin{aligned} |\mathcal{P}_L| &= |\mathcal{P}_{LU} \cup \mathcal{P}_{LD}| \\ &= |\mathcal{P}_{LU}| + |\mathcal{P}_{LD}| - |\mathcal{P}_{LUD}| \\ &= M_{\mathcal{P}} + M_{\mathcal{P}} - |\mathcal{P}_{LUD}| \\ &\leq 2M_{\mathcal{P}}. \end{aligned}$$

This contradicts the fact that $\mathcal{P}_L$ contains at least $N_{\mathcal{P}}$ points. Thus this case is also not possible.

Case 4: $U \cap D \subset \overline{L}$

This is the only permissible case. ∎

Hence proved(Fig. 33). ∎

The above lemmas have the consequence that either $U \cap D \subset L$ or $U \cap D \subset \overline{L}$, and similarly either $L \cap R \subset U$ or $L \cap R \subset \overline{U}$. Now we can prove the following theorem.

Theorem 4.6 *There are at least $M_{\mathcal{P}}$ points in $\mathcal{P}_{RD}$.*

Proof: For the proof, we again consider all the possible relative positions of the halfplanes L, U, R and D. The following cases arise, depending on which of the four quadrants formed by the boundaries of U and L contains p_{RD}.

Case 1: $p_{RD} \in U \cap L$ (Fig. 34)

In this case $D \cap U \cap L$ and $R \cap U \cap L$ are non empty. Therefore, from Lemma 4.4,

$$U \cap D \subset L \quad \text{and} \quad L \cap R \subset U.$$

Using Corollary 4.1 these respectively imply that

$$\overline{U} \cap R \subset \overline{L} \quad \text{and} \quad \overline{L} \cap D \subset \overline{U}.$$

Let x be a point of $R \cap D$. Then x lies in exactly one of the sets, $U \cap L$, $U \cap \overline{L}$, $\overline{U} \cap L$ or $\overline{U} \cap \overline{L}$. Since $\overline{U} \cap R \subset \overline{L}$ therefore $x \notin \overline{U} \cap L$. Similarly $x \notin U \cap \overline{L}$. Now $R \cap D$ is convex and there is a point p_{RD} that lies in

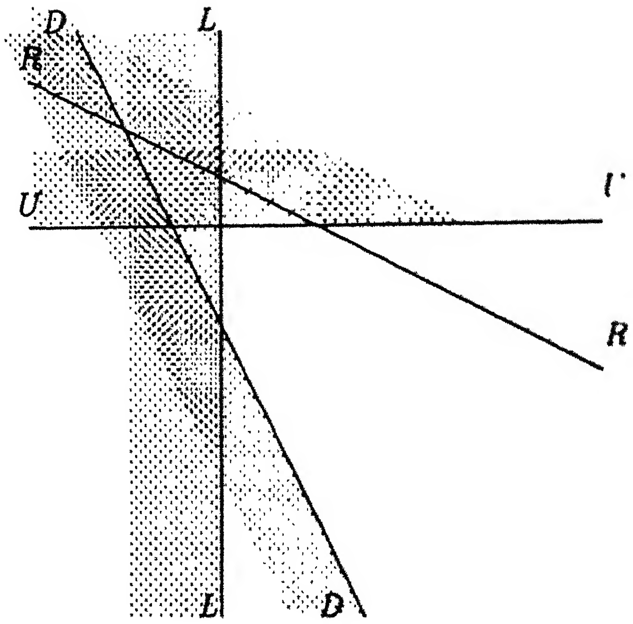

Figure 34: $p_{RD} \in U \cap L$

$U \cap L$, therefore x does not lie in $\overline{U} \cap \overline{L}$ either because a convex region cannot intersect only the opposite quadrants of a pair of lines. Thus

$$R \cap D \subset U \cap L,$$

and hence, by Corollary 4.1,

$$\overline{U} \cap R \subset \overline{D} \quad \text{and} \quad \overline{L} \cap D \subset \overline{R}.$$

Now,

$$\begin{aligned}
\mathcal{P} &= \mathcal{P}_L \cup \mathcal{P}_{\overline{L}} \\
&= \mathcal{P}_L \cup \mathcal{P}_{\overline{L}U} \cup \mathcal{P}_{\overline{L}\overline{U}} \\
&= \mathcal{P}_L \cup \mathcal{P}_{\overline{L}U} \cup \mathcal{P}_{\overline{L}\overline{U}R} \cup \mathcal{P}_{\overline{L}\overline{U}\overline{R}} \\
&= \mathcal{P}_L \cup \mathcal{P}_{\overline{L}U} \cup \mathcal{P}_{\overline{U}R} \cup \mathcal{P}_{\overline{L}\overline{U}\overline{R}} && \text{since } \overline{U} \cap R \subset \overline{L} \\
&= \mathcal{P}_L \cup \mathcal{P}_{\overline{L}U} \cup \mathcal{P}_{\overline{U}R} \cup \mathcal{P}_{\overline{L}\overline{U}\overline{R}D} \cup \mathcal{P}_{\overline{L}\overline{U}\overline{R}\overline{D}} \\
&= \mathcal{P}_L \cup \mathcal{P}_{\overline{L}U} \cup \mathcal{P}_{\overline{U}R} \cup \mathcal{P}_{\overline{L}D} \cup \mathcal{P}_{\overline{L}\overline{U}\overline{R}\overline{D}} && \text{since } \overline{L} \cap D \subset \overline{U} \text{ and } \overline{L} \cap D \subset \overline{R}.
\end{aligned}$$

As these are disjoint sets

$$|\mathcal{P}| \;\geq\; (N_{\mathcal{P}}) + (N_{\mathcal{P}} - M_{\mathcal{P}}) + (N_{\mathcal{P}} - M_{\mathcal{P}}) + (N_{\mathcal{P}} - M_{\mathcal{P}}) + |\mathcal{P}_{\overline{L}\overline{U}\overline{R}\overline{D}}|$$

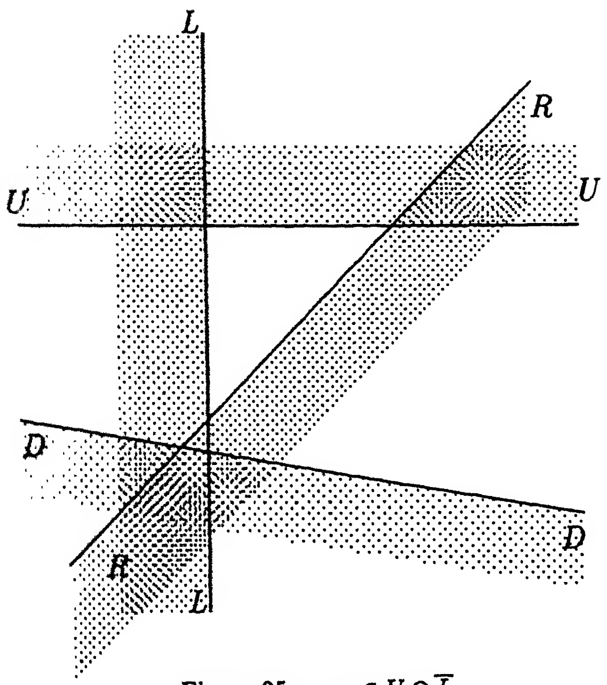

Figure 35: $p_{RD} \in U \cap \overline{L}$

$$\begin{aligned} &\geq \lceil|\mathcal{P}|/3\rceil + 3\lceil|\mathcal{P}|/4\rceil + |\mathcal{P}_{\overline{L}\,\overline{U}\,\overline{R}\,\overline{D}}| \\ &> |\mathcal{P}|, \end{aligned}$$

which is a contradiction. Hence this case does not occur.

Case 2: $p_{RD} \in \overline{U} \cap L$ (Fig. 35)

In this case $R \cap L \cap \overline{U} \neq \emptyset$ therefore $R \cap L \subset \overline{U}$ (Lemma 4.5). We will have to consider four subcases.

1. $R \cap D \subset L$
2. $\overline{R} \cap \overline{D} \subset L$

   Since $R \cap D \subset L$ therefore R and D contain the "opposite ends" of the boundary of L. Since $R \cap L \subset \overline{U}$ therefore from Corollary 4.1 R and U contain the "opposite ends" of the boundary of L. This implies that D and U contain the "same ends" of the boundary of L, which is not possible by the construction of D.
3. $\overline{R} \cap D \subset L$

By Lemma 4.3 this implies,

$$\overline{L} \cap D \subset R.$$

Therefore,

$$\begin{aligned} \mathcal{P}_{RD} &= \mathcal{P}_{RDL} \cup \mathcal{P}_{RD\overline{L}} \\ &= \mathcal{P}_{RDL} \cup \mathcal{P}_{D\overline{L}}. \end{aligned}$$

Thus

$$\begin{aligned} |\mathcal{P}_{RD}| &\geq |\mathcal{P}_{D\overline{L}}| \\ &\geq N_{\mathcal{P}} - M_{\mathcal{P}}. \end{aligned}$$

4. $R \cap \overline{D} \subset L$

   This implies

   $$R \cap U \subset D.$$

   Otherwise consider a point $x \in R \cap U \cap \overline{D}$,

   $$x \in R \cap U \cap \overline{D} \Rightarrow x \in R \cap \overline{D} \Rightarrow x \in L.$$

   Again,

   $$x \in R \Rightarrow x \in R \cap L \Rightarrow x \in \overline{U}.$$

   which is a contradiction.

   Therefore,

   $$\begin{aligned} \mathcal{P}_{RD} &= \mathcal{P}_{RDU} \cup \mathcal{P}_{RD\overline{U}} \\ &= \mathcal{P}_{RU} \cup \mathcal{P}_{RD\overline{U}}. \end{aligned}$$

   Hence,

   $$\begin{aligned} |\mathcal{P}_{RD}| &\geq |\mathcal{P}_{RU}| \\ &\geq M_{\mathcal{P}}. \end{aligned}$$

Since $p_{RD} \in L$ so these four cases exhaust all the possibilities.

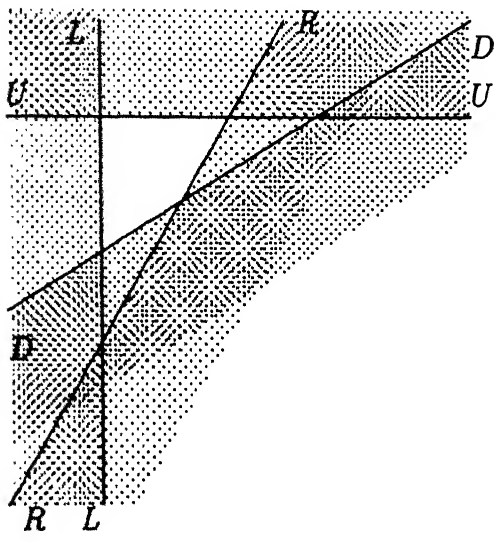

Figure 36: $p_{RD} \in \overline{U} \cap \overline{L}$

Case 3: $p_{RD} \in U \cap \overline{L}$

This case is similar to the previous one.

Case 4: $p_{RD} \in \overline{U} \cap \overline{L}$ (Fig. 36)

Again four different subcases are possible.

1. $R \cap L \subset U$ and $U \cap D \subset L$

   Consider a point $x \in R \cap D$. If $x \in L$ then $x \in U$ by $R \cap L \subset U$. If $x \in \overline{L}$ then $x \in \overline{U}$ by $U \cap D \subset L$. Since $R \cap D$ is convex, it cannot intersect exactly two opposite quadrants of a pair of lines. As there is a point p_{RD} in $\overline{U} \cap \overline{L}$ hence $R \cap D \subset \overline{U} \cap \overline{L}$.

   Assume that $|\mathcal{P}_{RD}| = m$. Then,

$$\begin{aligned}\mathcal{P}_{\overline{U}\overline{L}} &= \mathcal{P}_{\overline{U}\overline{L}RD} \cup \mathcal{P}_{\overline{U}\overline{L}\overline{R}D} \cup \mathcal{P}_{\overline{U}\overline{L}R\overline{D}} \cup \mathcal{P}_{\overline{U}\overline{L}\overline{R}\overline{D}} \\ &= \mathcal{P}_{\overline{U}RD} \cup \mathcal{P}_{\overline{L}\overline{R}D} \cup \mathcal{P}_{\overline{U}R\overline{D}} \cup \mathcal{P}_{\overline{U}\overline{L}\overline{R}\overline{D}}, \quad \text{since } \overline{U} \cap R \subset \overline{L} \text{ and} \\ &= \mathcal{P}_{RD} \cup \mathcal{P}_{\overline{L}\overline{R}D} \cup \mathcal{P}_{\overline{U}R\overline{D}} \cup \mathcal{P}_{\overline{U}\overline{L}\overline{R}\overline{D}}, \quad \text{since } R \cap D \subset \overline{U}\end{aligned}$$

   As the sets on the left of the last equality are disjoint sets,

$$|\mathcal{P}_{\overline{U}\overline{L}}| = |\mathcal{P}_{RD}| + |\mathcal{P}_{\overline{L}\overline{R}D}| + |\mathcal{P}_{\overline{U}R\overline{D}}| + |\mathcal{P}_{\overline{U}\overline{L}\overline{R}\overline{D}}|.$$

Substituting the values of the different terms, we get,

$$\begin{aligned}|\mathcal{P}| - 2N_{\mathcal{P}} + M_{\mathcal{P}} &\geq m + (N_{\mathcal{P}} - M_{\mathcal{P}} - m) + (N_{\mathcal{P}} - M_{\mathcal{P}} - m) + |\mathcal{P}_{\overline{U}L\overline{R}\overline{D}}| \\ |\mathcal{P}_{\overline{U}L\overline{R}\overline{D}}| &\leq m + |\mathcal{P}| - 4N_{\mathcal{P}} + 3M_{\mathcal{P}} \\ &\leq m - M_{\mathcal{P}} - 4\lceil|\mathcal{P}|/4\rceil + |\mathcal{P}|.\end{aligned}$$

Since the size of the set $\mathcal{P}_{\overline{U}L\overline{R}\overline{D}}$ is non-negative,

$$m \geq M_{\mathcal{P}}.$$

2. $R \cap L \subset U$ and $U \cap D \subset \overline{L}$

Since $p_{RD} \in \overline{U} \cap \overline{L}$ therefore p_{RD} lies in $\overline{U}$. There are four different possibilities.

- $R \cap D \subset \overline{U}$
- $\overline{R} \cap \overline{D} \subset \overline{U}$

  By a similar argument as in case 2 we can show that the above two cases are not possible.
- $R \cap \overline{D} \subset \overline{U}$

  Consider a point $x \in R \cap L$.

$$\begin{aligned}R \cap L \subset U &\Rightarrow x \in U \\ R \cap \overline{D} \subset \overline{U} &\Rightarrow x \in D \\ U \cap D \subset \overline{L} &\Rightarrow x \in \overline{L} \\ &\Rightarrow x \in L \cap \overline{L},\end{aligned}$$

  which is a contradiction. Hence $R \cap L = \emptyset$. Therefore $R \cap L \subset \overline{U}$ and $U \cap D \subset \overline{L}$ — a subcase we shall consider later.
- $\overline{R} \cap D \subset \overline{U}$

  Let $|\mathcal{P}_{RD}| = m$. We compute $|\mathcal{P}_{\overline{D}R\overline{U}\overline{L}}|$.

  Now,

$$\begin{aligned}\mathcal{P} &= \mathcal{P}_D \cup \mathcal{P}_{\overline{D}} \\ &= \mathcal{P}_D \cup \mathcal{P}_{\overline{D}R} \cup \mathcal{P}_{\overline{D}\overline{R}}\end{aligned}$$

$$
\begin{aligned}
&= \mathcal{P}_D \cup \mathcal{P}_{\overline{D}R} \cup \mathcal{P}_{\overline{D}\,\overline{R}U} \cup \mathcal{P}_{\overline{D}R\overline{U}} \\
&= \mathcal{P}_D \cup \mathcal{P}_{\overline{D}R} \cup \mathcal{P}_{\overline{R}U} \cup \mathcal{P}_{\overline{D}R\overline{U}L} \cup \mathcal{P}_{\overline{D}R\overline{U}\overline{L}}, \qquad \text{since } \overline{R} \cap U \subset \overline{D} \\
&= \mathcal{P}_D \cup \mathcal{P}_{\overline{D}R} \cup \mathcal{P}_{\overline{R}U} \cup \mathcal{P}_{\overline{D}\,\overline{U}L} \cup \mathcal{P}_{\overline{D}R\overline{U}\overline{L}},
\end{aligned}
$$

since $L \cap \overline{U} \subset \overline{R}$.
Substituting the values of different terms in the above equation we get

$$
\begin{aligned}
|\mathcal{P}| &\geq N_{\mathcal{P}} + N_{\mathcal{P}} - m + N_{\mathcal{P}} - M_{\mathcal{P}} + N_{\mathcal{P}} - 2M_{\mathcal{P}} + |\mathcal{P}_{\overline{D}R\overline{U}\overline{L}}| \\
|\mathcal{P}_{\overline{D}R\overline{U}\overline{L}}| &\leq m - 4N_{\mathcal{P}} + 3M_{\mathcal{P}} + |\mathcal{P}| \\
&\leq m - M_{\mathcal{P}} - 4\lceil |\mathcal{P}|/4 \rceil + |\mathcal{P}|.
\end{aligned}
$$

Since the size of the set $\mathcal{P}_{\overline{D}R\overline{U}\overline{L}}$ is non negative therefore,

$$m \geq M_{\mathcal{P}}.$$

3. $R \cap L \subset \overline{U}$ and $U \cap D \subset L$

   This case can be dealt with in a similar way as above.

4. $R \cap L \subset \overline{U}$ and $U \cap D \subset \overline{L}$

   Since $p_{RD} \in \overline{U}$ here also we shall consider four different cases.

   - $R \cap D \subset \overline{U}$
   - $\overline{R} \cap \overline{D} \subset \overline{U}$

     These cases do not occur for the same reasons as discussed above.
   - $R \cap \overline{D} \subset \overline{U}$

     From Corollary 4.1,

     $$R \cap U \subset R \cap D.$$

     Since $R \cap U$ contains $M_{\mathcal{P}}$ points of $\mathcal{P}$,

     $$
     \begin{aligned}
     |\mathcal{P}_{RD}| &\geq |\mathcal{P}_{RU}| \\
     &\geq M_{\mathcal{P}}
     \end{aligned}
     $$

- $\overline{R} \cap D \subset \overline{U}$

  We need only consider the case $R \cap \overline{D} \subset \overline{L}$ since other cases, viz. $R \cap D \subset \overline{L}$, $\overline{R} \cap D \subset \overline{L}$ and $\overline{R} \cap \overline{D} \subset \overline{L}$, can be dealt symmetrically as above (actually they leads to a contradiction when considered along with $\overline{R} \cap D \subset \overline{U}$). Let $\mathcal{P}_{RD}$ contain m points. As before we compute the number of points in $\mathcal{P}_{\overline{D}\overline{R}\overline{U}\overline{L}}$.

$$\begin{aligned}
\mathcal{P} &= \mathcal{P}_D \cup \mathcal{P}_{\overline{D}} \\
&= \mathcal{P}_D \cup \mathcal{P}_{\overline{D}R} \cup \mathcal{P}_{\overline{D}\overline{R}} \\
&= \mathcal{P}_D \cup \mathcal{P}_{\overline{D}R} \cup \mathcal{P}_{\overline{D}\overline{R}U} \cup \mathcal{P}_{\overline{D}\overline{R}\overline{U}} \\
&= \mathcal{P}_D \cup \mathcal{P}_{\overline{D}R} \cup \mathcal{P}_{\overline{R}U} \cup \mathcal{P}_{\overline{D}\overline{R}\overline{U}L} \cup \mathcal{P}_{\overline{D}\overline{R}\overline{U}\overline{L}}, \qquad \text{since } \overline{R} \cap U \subset \overline{D} \\
&= \mathcal{P}_D \cup \mathcal{P}_{\overline{D}R} \cup \mathcal{P}_{\overline{R}U} \cup \mathcal{P}_{\overline{D}\overline{U}L} \cup \mathcal{P}_{\overline{D}\overline{R}\overline{U}\overline{L}},
\end{aligned}$$

  since $L \cap \overline{D} \subset \overline{R}$.

  Substituting the values of different terms in the above equation we get

$$\begin{aligned}
|\mathcal{P}| &\geq N_{\mathcal{P}} + N_{\mathcal{P}} - m + N_{\mathcal{P}} - M_{\mathcal{P}} + N_{\mathcal{P}} - 2M_{\mathcal{P}} + |\mathcal{P}_{\overline{D}\overline{R}\overline{U}\overline{L}}| \\
|\mathcal{P}_{\overline{D}\overline{R}\overline{U}\overline{L}}| &\leq m - 4N_{\mathcal{P}} + 3M_{\mathcal{P}} + |\mathcal{P}| \\
&\leq m - M_{\mathcal{P}} - 4\lceil |\mathcal{P}|/4 \rceil + |\mathcal{P}|.
\end{aligned}$$

  Since the size of the set $\mathcal{P}_{\overline{U}\overline{L}\overline{R}\overline{D}}$ is non negative therefore,

$$m \geq M_{\mathcal{P}}.$$

Thus theorem is proved for the last case $p_{RD} \in \overline{U} \cap \overline{L}$ also. ∎

nce proved. ∎

mbining the earlier theorems and lemmas we get the following result.

Theorem 4.7 *A point in the centre of a set can be computed in linear time.*

Proof: In each iteration at least $3M_{\mathcal{P}}$ ($\sim |\mathcal{P}|/4$) points are deleted.

If $T(n)$ is the running time of the algorithm for an input set of size n ($|\mathcal{P}| = n$), then it satisfies the following recurrence.

$$\begin{aligned} T(n) &\leq \max_{k \geq M_{\mathcal{P}}} T(n-3k) + O(n) \\ \Rightarrow \quad T(n) &\leq T(n - 3M_{\mathcal{P}}) + O(n) \end{aligned}$$

Since $T(n) = O(n)$ from the above recurrence relation, the claim of the theorem follows. ∎

4.6 Concluding Remarks

We have presented an optimal algorithm for computing a centrepoint of a finite set of points in the plane, thus providing one more example of the power and versatility of the prune and search paradigm.

It would be worth exploring how this speeds up algorithms which uses the centrepoint computation as a basic subroutine.

Chapter 5

Designing Algorithms Using Partial Sorting Networks

In this chapter we present a general technique that is a sequel to the parametric searching technique of Megiddo. The latter technique is used to design serial algorithms with the help of efficient parallel ones. However, these algorithms are not very efficient for the class of problems that use parallel sorting networks in their solutions. We modify this technique so that we obtain optimal linear time algorithms for some of these problems. We do this by synthesising prune and search technique and parametric search technique.

5.1 Introduction

The parametric searching of Megiddo is as follows — Let there be an efficient parallel algorithm for a problem $\mathcal{A}$ such that solution of $\mathcal{A}$ can be used in the solution of another problem $\mathcal{B}$. Then, in some cases, we get an efficient sequential algorithm for $\mathcal{B}$ by exploiting the efficient parallel mechanism of the parallel algorithm for $\mathcal{A}$. This technique has been applied to a wide variety of problems yielding efficient algorithms. In particular, we achieve good results for the parameterised problems that use parallel sorting algorithms in their solutions. We do this by replacing the evaluation of the parallel comparisons of an iteration in the parallel version by simultaneous resolution of these in the serial version. The running time of these algorithms is further improved by introduction of weights in the comparisons [8]. We then simultaneously evaluate at least a fraction of total weight of these in every iteration to design more efficient algorithms.

We further improve the running time of the above algorithms by introducing, wherever applicable, prune and search in these. We do this by seeking to compute the k-th largest element of the input set instead of seeking to sort it as in previous techniques. For this, we run the sorting algorithm on a given input for a few iterations and then prune the set. It can be easily seen that this approach is useful only where pruning is applicable.

This chapter is organised as follows. In section 5.2 we review a few definitions and concepts related to the AKS sorting networks. In section 5.3 we present a method of applying prune and search using these networks. In section 5.4 we optimally solve the problem that was first posed by Megiddo. In subsequent sections we apply this technique to linear programming problem, computation of ham sandwich cuts and computation of centre points in d-dimensions.

5.2 Preliminaries and Definitions

The AKS sorting network sorts a sequence of reals in $O(n \log n)$ comparisons such that in each parallel iteration $O(n)$ comparisons can be performed so that there are in total $O(\log n)$ iterations. A *comparison* operation in the network consists of comparing the contents of two registers and interchanging these if their specified order does not match.

We can view the process of sorting in the AKS sorting network as the movement of gisters in a complete binary tree of n leaves. The order of the leaf (topmost) nodes this tree corresponds to the ordered sequence of n numbers. All the registers are itially in the root (bottommost) node of the tree and in every iteration, most of e registers in a node move up whereas only a small number of registers move down. hen each of the registers reaches a leaf such that each leaf has only one register esent in it then the computation is said to be over and the elements in the leaves e in sorted order.

'e use the same notation as in [1] by Ajtai et al. Let $\mathcal{R}$ be a set of n registers, iich contain elements of the input set, in the AKS sorting network. Let the set registers assigned to a node t of the complete binary tree in α-th iteration be :noted by $S^\alpha(t)$. Thus, $S^0(t)$ is $\mathcal{R}$ if t is the root of tree, otherwise it is $\emptyset$ (empty t). The restriction of the function S^α to a fixed level of height h is denoted by $^{\alpha,h}$. S^α is defined in such a way that each of the nodes at any height, h, contain me number of registers. Let $N(S^{\alpha,h})$ be this number. We bound this number in e following theorem. We reassign the constants q_1, q_2 and c_1 used in the paper [1] q, Q and c respectively in the following discussion ($Q \ll q \ll 1/c \ll 1$).

heorem 5.1 ([1]) *The number of registers in any node at height h in α-th itera-on has the following properties:*

1. $N(S^{\alpha,h}) \leq q^{\alpha-h}2^{-\alpha}n$.
2. *if* $q^{\alpha-h}2^{-\alpha-1}n \leq c$ *then* $N(S^{\alpha,h}) = 0$ *otherwise* $N(S^{\alpha,h}) \geq q^{\alpha-h}2^{-\alpha-1}n$.

'e can see from the above theorem that the number of registers in any node in ·th iteration is bounded, both above and below, by terms that are in decreasing :ometric sequence. The AKS sorting network has another useful property. The dering of the elements after each iteration is an approximation of the sorted order the input set. Moreover, at any instant the number of the elements having a rge displacement is very small. We bound this number in the following theorem.

Let the *relative position* of a number x in a sequence be defined as the fraction of numbers in the sequence that are smaller than x. We also assume that the positions to right correspond to the larger elements.

Theorem 5.2 ([1]) *Let $G_\mu(t)$, where t is a node at height h ($h \leq \alpha$), be the set of registers in the nodes, right of t, at height h such that the relative positions of elements in these are smaller by at least μ than the relative position of t. Similarly, let $H_\mu(t)$ be the corresponding set of registers with larger elements on the left of t such that the relative positions of the elements contained in them are larger by μ than the relative position of t. Then,*

$$\left|G_{(2^m/M)}(t)\right| \leq q^{\alpha-h}2^{-\alpha}nQ^m$$

and, similarly,

$$\left|H_{(2^m/M)}(t)\right| \leq q^{\alpha-h}2^{-\alpha}nQ^m$$

where $M = N(S^{\alpha,h})$ and $m \geq 1$.

Now, we seek to apply the Megiddo's technique to optimisation problems with the aid of these theorems. In these, we frequently need to prune redundant hyperplanes in E^d. The multi dimensional prune and search procedure proposed by Megiddo is used for this purpose [22]. According to this, we can prune a fraction of the hyperplanes by locating the optimal point with respect to a finite set of query hyperplanes. The following theorem establishes the existence of these query hyperplanes.

Theorem 5.3 ([22]) *For any dimension d there exist constants $A = A(d)$ and $B = B(d)$, with $0 < B \leq 1/2$, such that A queries suffices to determine the position of optimal point relative to at least Bn of n hyperplanes in E^d, where*

$$A(d) = 2^{d-1}$$

and

$$B(d) = 2^{1-2^d}.$$

It is clear that this procedure of pruning hyperplanes is applicable only if there exists an oracle, say Ω, that takes linear time to determine the relative position of the optimal point with respect to a query hyperplane.

In the next section we discuss the pruning of elements in the input set.

5.3 Pruning Using the AKS Sorting Networks

Let S is a set of n elements. Then, there are altogether N $(= {}^nC_2)$ distinct comparisons possible between pairs of the elements of S. We assume that the output of any comparison depends on the value of a vector $\mathbf{x}$ i.e. it depends on the membership of $\mathbf{x}$ among disjoint subsets of the domain. In the problems we discuss, these are the regions determined by hyperplanes. We call these critical hyperplanes. If the optimal points $\mathbf{x}^*$ is known then the complete order of elements of S is known. The critical hyperplane of each unresolved comparison intersects the localised region. Furthermore, it is to be noted that localisation of $\mathbf{x}^*$ with respect to one of these hyperplanes as well as localisation with respect to $O(n)$ of these (only a fraction of these will be resolved at a time using multi dimensional prune and search) require linear time(Theorem 5.3).

We pose a hypothetical problem of computing the k-th largest element, s_k, of S at $\mathbf{x}^*$. What are the elements that can be pruned ? Clearly, these are the elements that are not the k-th largest element at $\mathbf{x}^*$. We formulate our problem as follows:

Subproblem 1 *How can we determine, in linear time, $O(n)$ elements of S that are not same as s_k given an oracle Ω that takes $O(n)$ time to answer the following query — which side of a given hyperplane does $\mathbf{x}^*$ lies?*

The real crux of the problem now is to identify the elements that have large displacement from s_k. For determination of these, we generalise the method of Lo and

Steiger in [18]. We determine a prismoid, whose base is the localised region, about the k-th largest element such that most of the elements do not intersect it. The elements that do not intersect this prismoid are always at a non zero finite relative displacement from s_k in the localised region and therefore, can be pruned.

5.3.1 What to Prune ?

Let S be a set of hyperplanes in E^d. Suppose we can localise $\mathbf{x}^*$ in a region J such that the total number of unresolved comparisons is only a small fraction of N. We construct an infinite prism with J as its base such that only few of the intersections of any pair of elements of S intersect J. Then we prune the hyperplanes by the following theorem.

Theorem 5.4 *Let $\mathcal{P}$ be the prismoid determined by $k-(d-1)\lceil \varepsilon n \rceil$ and $k+(d-1)\lceil \varepsilon n \rceil$ largest intercept on each vertex of localised region (which is a simplex). If there are less than ${}^dC_2\lceil \varepsilon n \rceil^2$ unresolved comparisons in the localised region then we can prune approximately $(n-d\lfloor \sqrt{2}\varepsilon n \rfloor)$ elements in the computation of k-th largest element.*

Proof: Each unresolved comparison corresponds to the intersection of a pair of hyperplanes in S which also intersects the infinite prism of which $\mathcal{P}$ is a part.

We compute the minimum number of these intersections needed so that the surface formed by k-th largest elements (k-level) may lie above or below the prismoid at an interior point p. We only analyse the case when the surface lies above $\mathcal{P}$ at p.

Let us consider the following worst case configuration. In this configuration, each of the hyperplanes that is above the k-level and below the $k+(d-1)\lceil \varepsilon n \rceil$-level at any of the vertices v_i, with $0 \leq i \leq d$ of the localised region is above the prismoid at p. Let the set of these hyperplanes be H. Also, in the minimal case, no hyperplane that is below the k-level at any of the v_i's is above the prismoid at any point. Otherwise, the number of intersections in J increases. Similarly, no hyperplane, that is not in H but which is above the $k+(d-1)\lceil \varepsilon n \rceil$-level at any of the v_i's, intersects prismoid. Furthermore, none of these hyperplanes intersects any other hyperplane over J.

Now, the hyperplanes in H in this configuration can be divided into several equivalence classes defined as follows. If two hyperplanes intersect or do not intersect the prismoid at the same vertices then these belong to same equivalence class. Each equivalence can be represented by a set of vertices of the base, at which the hyperplanes of this class intersect $\mathcal{P}$. We call these as *intersecting vertices* of an equivalence class. A hyperplane in one equivalence class intersects a hyperplane of another equivalence class if their intersecting vertices are not related by the relations of either subset or superset.

If there are d equivalence classes of size $\lceil \varepsilon n \rceil$ such that each has only one intersecting vertex then the number of intersections above J is minimum. The count the number of intersections in this configuration is ${}^dC_2 \lceil \varepsilon n \rceil^2$. If the number of intersections is less than this then the k-level will lie inside the prismoid.

We can prune those hyperplanes which are above or below the prismoid at all the vertices v_i. The maximum number of hyperplanes that will intersect the prismoid is when the equivalence classes are chosen as above and the intersections are divided equally between the hyperplanes both above and below the k-level. Hence, we can prune the rest of approximately $(n - d\lfloor \sqrt{2}\varepsilon n \rfloor)$ hyperplanes in $\mathcal{S}$. ∎

We can prove another similar theorem.

Theorem 5.5 *If there are less than ${}^dC_2 \lceil \varepsilon n \rceil^2$ unresolved comparisons in the feasible region then the prismoid is formed by the largest (smallest) and $(d-1)\lceil \varepsilon n \rceil$ largest (smallest) intercept on each vertex of feasible region (which is a simplex) and we can prune $(n - d\lceil \varepsilon n \rceil)$ elements in the computation of the largest (smallest) element.*

To use prismoid method we need to ensure that number of unresolved comparisons in the feasible region should only be a small fraction of N. Then we can guaruntee the existence of a non zero fraction of prunable elements. In next section, we compute the minimum number of iterations of AKS sorting network needed to ensure this.

5.3.2 Why can we Prune ?

We use the properties of AKS sorting network stated in the previous section to prove that the number of unresolved comparisons in α-th iteration is bounded by a number that decreases in a geometrical progression with α.

Theorem 5.6 *The number of unresolved comparisons in the AKS sorting network in the α-th iteration, out of a maximum of ${}^nC_2$ ones, is bounded from above by*

$$\left(\left(\frac{Q}{1-q}+\frac{1}{2-q^2}\right)2^{-\alpha}+\frac{2}{c}\right)n^2.$$

Proof: Let us first count the number of unresolved comparisons of an element x in a register present in a node at height h in α-th iteration. Let μ be $2/c$ and t be that node which has the same relative position as x. Then

$$\begin{aligned}
2^m/M &\leq \mu,\\
2^m &\leq \mu M\\
&\leq 2M/c\\
&\leq 2q^{\alpha-h}2^{-\alpha-1}n/c\\
&\leq q^{\alpha-h}2^{-\alpha}n/c.
\end{aligned}$$

Since $q^{\alpha-h}2^{-\alpha-1}n \geq c$, therefore, $m = 1$ satisfies the above inequality.

Thus, the number of unresolved comparisons of elements at height h with a larger relative displacement than μ from x is bounded by

$$|G_\mu(t)| + |H_\mu(t)| \leq 2 \cdot q^{\alpha-h}2^{-\alpha}nQ.$$

This quantity when summed over each level of the computation tree is at most

$$\begin{aligned}&\sum_{0<h\leq\alpha} 2q^{\alpha-h}2^{-\alpha}nQ\\ \leq\ & 2(\frac{1}{1-q})2^{-\alpha}nQ.\end{aligned}$$

The number of remaining unresolved comparisons of x with elements having relative displacement smaller than μ is at most $2\mu = 4n/c$. We take half of this in the total as each of these unresolved comparisons is counted twice in the sum. Further, we also add unresolved comparisons corresponding to any two registers in the same node to the computed total. Therefore, number of total unresolved comparisons is at most

$$\begin{aligned}&\left((\frac{1}{1-q})2^{-\alpha}nQ + 2n/c\right)n + \sum_{0<h\leq\alpha} 2^h N(S^{\alpha,h})^2/2\\ \leq\ & \left((\frac{1}{1-q})2^{-\alpha}nQ + 2n/c\right)n + \frac{2^{-\alpha}n^2}{2-q^2}\\ \leq\ & \left(\left(\frac{Q}{1-q} + \frac{1}{2-q^2}\right)2^{-\alpha} + \frac{2}{c}\right)n^2.\end{aligned}$$

Hence proved. ∎

We can make this fraction as small as we need by choosing α, q, Q, c appropriately. We can thus bound the number of unresolved comparisons in each iteration. But there still remains an algorithmic problem. Resolving all comparisons of even only a first few iterations of the AKS sorting network requires $O(\log n)$ steps of Megiddo's multi dimensional prune and search. This will take at least $O(n \log n)$ time. However, this is not agreeable. So, in order to make the pruning procedure linear, we have to leave another small fraction, say $\delta/2$, of comparisons unresolved. We observe that this adds at most $2(\delta n/2)n$ in the above quantity i.e. two elements for each unresolved comparison (and these may remain uncompared with all the other elements). Then, the total number of unresolved comparisons is

$$\left(\left(\frac{2Q}{1-q} + \frac{1}{2-q^2}\right)2^{-\alpha} + \frac{2}{c} + \delta\right)n^2,$$

where δ is also chosen appropriately. Now, we can apply Megiddo's multi dimensional prune and search on the critical hyperplanes until only $\delta n/2$ comparisons of the total comparisons of first α iterations are left unresolved. Clearly, this takes linear time.

The method of application of the technique will become more clear in section 5.4 when we apply our technique to a few problems.

5.3.3 How to Prune ?

We can view AKS sorting network as having $O(\log n)$ iterations with at most $n/2$ comparators in every stage. We give a weight of 4^{-j} to each comparator at a depth j as in [8]. We choose the values of q, Q, c, δ, α and ε appropriately and assign weights to comparators of first α stages only. The construction of first α stages of AKS sorting network takes linear time. We give details of the algorithm to compute $\mathbf{x}^*$ below.

Algorithm 5 Prune And Search on AKS Sorting Network

```
Input: $S$
begin
    do
        Let $C$ be the set of unresolved comparisons
          and $W$ its total weight
        do
            Solve $(d-1) \cdot A(d-1)$ queries using $\Omega$ ( Theorem 5.3)
            Resolve at least $B(d-1)$ comparisons
            Update $C$ and $W$ appropriately
        while $|C| \geq \delta n/2$
        enddo
        Prune the hyperplanes in $S$ ( Theorem 5.4)
        Update $k$ and $S$ accordingly
    while $S$ is modified
    enddo
    Apply any brute force method to compute $x^*$
end ∎
```

In the above algorithm $(d-1)A(d-1)$ queries are put to the oracle because we need to ensure that the base of the prismoid, where x^* is located, be a simplex. So, whenever a query hyperplane intersect the base, we make at most $d-2$ additional queries to again make it a simplex.

It can be easily seen that this algorithm runs in linear time if the time complexity of oracle is $O(|S|)$.

5.4 Intersection of Median of Straight Lines with a Given Line

Let there be n non-vertical lines l_i, where $i = 1, \ldots, n$ in a Euclidean plane. Each of these lines is represented by a linear equation $y = f_i(x)$. The *median level* is defined as a function, F, such that $F(a)$ is the ordinate of $\lfloor n/2 \rfloor$-th intersection of the lines with the vertical line $x = a$. We can evaluate $F(x)$ for any x in linear time by computing median of $f_i(x)$'s.

Let us compute intersection of median level and a given line l, which is represented by $y = f(x)$. We are guarunteed a solution if the slope of this line is not the median slope if n is odd; does not lie between the $\lfloor n/2 \rfloor$-th and $\lfloor n/2+1 \rfloor$-th slope if n is even.

For the sake of simplicity we assume that all the lines have positive slope and the line l is the x-axis. Then we have to determine x^* such that $F(x^*) = f(x^*) = y^* = 0$.

At $x \rightarrow -\infty$ and at $x \rightarrow +\infty$ function $F(x)$ is same as the line of median slope. Therefore at $x \rightarrow +\infty$ it is above the x-axis and at $x \rightarrow -\infty$ it is below x-axis. Since function $F(x)$ is continuous, the equation $F(x) = 0$ has at least one solution. This problem was first posed by N. Megiddo to motivate the technique of parametric searching.

In the next subsections we briefly review how Megiddo and Cole applied their techniques to solve this problem.

5.4.1 An Algorithm by Megiddo that Motivates the Technique of Parametric Searching

We first make an important observation. The order of lines changes only at intersection points of a pair of lines. Hence, intersection points play an important role in the evaluation of x^*.

Let x_{ij} be the abscissae of the intersection point of lines l_i and l_j.

We can compute x^* naively as follows. First we identify intersection points of each pair of lines. Then we search for two values x^1 and x^2 such that $F(x^1) \leq 0 \leq F(x^2)$ and there is no other x_{ij} in the open interval (x^1, x^2). We search for such an interval by employing binary search. This requires $O(\log n)$ F-evaluations and median computation of subsets of x_{ij} whose cardinalities are $N, N/2, N/4, \ldots$ points where $N = {}^nC_2$. Thus, this algorithm runs in $O(n^2)$ time which is dominated by the evaluation of all the intersection points.

Now we employ Megiddo's technique to design an algorithm that runs in $O(n \log^2 n)$. We evaluate $F(x^*)$ with x^* not specified by computing the median of f_i's. The median-finding algorithm compares values of function $f_i(x^*)$ and $f_j(x^*)$ for different i's and j's. The outcome of such a comparison changes only at intersection point x_{ij}. Hence, if $F(x_{ij}) > 0$ then $x^* < x_{ij}$; if $F(x_{ij}) < 0$ then $x^* > x_{ij}$; and if $F(x_{ij} = 0$

then $x^* = x_{ij}$. Once we know the outcome of a comparison we proceed with the evaluation of median. Now the value of x^* is restricted in the interval $(-\infty, x_{ij})$ or $(x_{ij}, +\infty)$. At the end of median-finding algorithm we know the median line f_i at $x = x^*$. The intersection of this line with x-axis yields x^*. Clearly, this algorithm is also $O(n^2)$. Since the above algorithm is no better then the previous one we try another approach using AKS sorting networks. The AKS Sorting network employs $P = O(n)$ processors in every stage and sorts its input in $O(\log n)$ time. Instead of evaluating function $F(x)$ at each of the x_{ij}'s, we first wait for all the P values in a parallel iteration to be computed. Then we sort these P values and do a binary search to find the interval which contains x^*. Then, we resolve all the comparisons of this parallel iteration. When all the iterations of AKS sorting network are completed then the lines are sorted at x^* and computation of x^* is easy.

Since each stage takes $O(n \log n)$ time and there are $O(\log n)$ iterations, so the overall running time of this algorithm is $O(n \log^2 n)$.

5.4.2 Algorithm Using Slowed Down Sorting Networks

The above algorithm was further improved by Cole using slowed down sorting networks.

A brief outline of his technique is as follows. A comparator in the sorting network is either active or inactive. An active comparator has its inputs determined but the outputs not determined. The other comparators are inactive. An active comparator at depth d in the network is assigned a weight $1/4^d$. The intersection point of pair of its input, x_{ij}, is also assigned the same weight.

In Megiddo's algorithm we exhausted all the x_{ij}'s of each iteration before going to the next iteration. Here we simultaneously deal will all the x_{ij}'s of the active comparators with their respective weights. Accordingly, we evaluate F at weighted median of x_{ij}'s in the place of median, in the search of interval that contains

x^*. Thus, we resolve comparisons in either the left half, $(-\infty, x_m]$, or the right half, $[x_m, \infty)$ of the weighted median point x_m. These comparisons with weight approximately half the total weight become inactive and are replaced by active comparisons not more that one-fourth of the total weight as new weight. We reiterate these steps until we sort the lines at x^*. We claim the following.

Theorem 5.7 ([8]) *At the completion of kth iteration the active weight is bounded by* $(3/4)^k n/2$, *for* $k \geq 0$.

Theorem 5.8 ([8]) *For* $k \geq 5(d + 1/2 \log n)$, *after kth iteration there are no active comparators at depth* d, *where* $d \geq 0$.

Theorem 5.7 guaruntees that in every iteration we drop a finite fraction of total weight whereas Theorem 5.8 guarantees that the algorithm terminates in $O(\log n)$ steps. Therefore, after $O(\log n)$ steps we know the median of f_i's at x^* and hence x^*. The running time of this algorithm is $O(n \log n)$.

5.4.3 Application of Prune and Search Technique

Now we apply prune and search technique to this problem. Let us refer to Algorithm 5 presented in the previous section. The input of this algorithm is set of lines in the plane. We do the following:

- We prune the lines which are not the k-th largest lines (initially $k = \lfloor n/2 \rfloor$) at x^* and which lie above or below the trapezium given by Theorem 5.4.
- When some of the lines are pruned we update value of k.
- The oracle Ω is constructed by evaluating $F(p)$ at $x = p$ and determining position of x^* with respect to p with the help of sign of $F(p)$.

This algorithm runs in linear time which is optimal.

5.5 Linear Programming

The next problem that we solve is the linear programming problem in fixed dimensions. The linear programming problem is as follows.

$$\begin{array}{rl} \textbf{Minimise} & z \\ \textbf{Subject to} & \\ & \langle c_i, x\rangle \;\leq\; z, \quad 1 \leq i \leq n \end{array}$$

Each constraint defines a halfspace in a $d+1$ dimensional space. We try to compute the solution point x^*. We again apply the Algorithm 5 for computing the k-th largest hyperplane at x^*.

We construct the oracle Ω for this problem as follows. We recursively compute the optimum value constrained on the query hyperplane H. We can compute the relative position of the optimal point with respect to H using the gradient of largest constraint at the constrained optimal point on H. The base case in the recursion is a point P corresponding to 0-th dimension where the solution is trivial.

We sort the constraints and choose the largest constraint at x^*. Whenever a comparison is resolved the smaller hyperplane is pruned. Since the highest constraint is to be computed, Theorem 5.5 is applicable. This algorithm runs in linear time though it is not as efficient as that of Megiddo because of the large constants involved in the AKS sorting networks.

5.6 Ham-sandwich Cuts

Let A_i, where $1 \leq i \leq d$, be d finite sets of points in $\Re^d$. The ham-sandwich cut of these is a hyperplane such that it bisects each of these sets. To compute this cut we dualise the problem as usual. In the dual space the sets A_i transform to d sets of hyperplanes. We define this transformation as follows.

Primal Space	$y = \langle \vec{m}, \vec{x} \rangle + c$	$(\vec{x}, y)$
Dual Space	$(-\vec{m}, c)$	$Y = \langle \vec{x}, \vec{X} \rangle + y$

It should be noted that the above transformation conserves the incidence relationships.

A ham-sandwich cut transforms to the point, say h, such that in each set there are exactly half the total number of hyperplanes that are above h. Consider the problem of computing h. If the median hyperplanes of each set at the point h are known then h can be computed by computing the intersection point of these. Hence, we do the following in the Algorithm 5. The elements that are to be sorted are the above hyperplanes and the ordering of the hyperplanes is to be done at h. A comparison in the sorting network would produce a "ridge". We need to localise h with respect to these "ridges". The comparisons corresponding to those ridges for which h is localised get resolved.

In the oracle Ω we need to know which side of a given hyperplane H does the point h lie. For this we use the method of Lo, Matousek and Steiger[17]. This provides us a method to determine if h lies inside a given infinite prism. This oracle Ω first computes the median level of one of the sets on each vertical side of prism and then counts the total number of intersections below this level of the median levels of other $d-1$ sets. If this number is odd then h lies inside the prism. The complexity of this algorithm is bounded by the complexity of constructing a k^{th}-level in the $d-1$ dimensional plane. So we get the same time bounds as in [17]. The linearly separated case in E^3 can also be solved similarly in linear time.

5.7 Centrepoints in d-dimensions

In this section we discuss the problem of computing centrepoints of sets of points in d-dimensional space. Suppose S is a set of n points in $\Re^d$. The *centre* of S is union of all points p such that any closed halfspace containing p contains at least $\lceil n/(d+1) \rceil$ points. We obtain a weaker type of centre, *approximate* or *ε-centre* if we decrease the lower bound in the above definition of centre to $\lceil n(1-\varepsilon)/(d+1) \rceil$, where $0 < \varepsilon \leq 1$.

We first discuss the method of computing an approximate or ε-centre of set S, where $0 < \varepsilon \lll 1$.

To compute an ε-centrepoint we dualise the centrepoint problem. However, as opposed to ham-sandwich cut algorithm in the previous section, here we need to consider both the primal space as well as the dual space, in the algorithm. In particular, we search the centre in the dual space and prune the points in the primal space. As before we employ the technique by applying Algorithm 5.

A centrepoint in the dual space will be a hyperplane (a *centre-hyperplane*) such that in every vertical line there would be at least $\lceil n/(d+1) \rceil$ intersections of given hyperplanes above and below the intersection of centre-hyperplane. The dual of centre is union of such hyperplanes. The ε-centre-hyperplane is defined similarly.

The elements to be sorted are the dual hyperplanes of points in S. We apply sorting on these hyperplanes. Since We do not execute the AKS sorting network to its completion, we can be vague about the sorting order and so we do not specify the exact point at which the sorting is taking place. We apply the algorithm just to partition the space such that each partition contains only small number of related unresolved comparisons. In the absence of localisation the oracle Ω is not needed here.

Whenever a hyperplane is queried in the Algorithm 5, we process the next steps allowing all the possible outcomes of the constrained search of the oracle Ω. In this way whole of the region is partitioned into several sets each containing only small number of intersection with "ridges" related to unresolved comparisons. We iterate until the maximum of these numbers drops below $\varepsilon^2 \cdot {}^d C_2 n^2$. Then we apply the prismoid method in each partition to compute 4ε-centre. It can be seen that the region determined by $\lceil n/(d+1) \rceil$ lowest and highest levels is an 4ε-centre. This is the consequence of Theorem 5.4. Clearly, each face of 4ε-centre contains exactly $\lceil n/(d+1) \rceil$ points.

Since we are not searching for a solution point, we do not prune points as usual. We provide another method of pruning below. In the earlier chapter we saw how pruning can be done in the plane for collection of three and four points. These points were respectively deleted and substituted by the Radon's point. In fact, it can be shown that $(d+1)r+1$ points in $\Re^d$, for any r, can be suitably substituted by their Tverberg's point (a generalisation of Radon's point) when certain conditions are met. Unfortunately, this also can not be applied in our case. To prune the points, we conjecture the following[33].

Conjecture 5.1 *If there are $(d+1)r+q$, where $0 \leq q \leq r$ points in E^d such that each r-set is contained in some halfspace containing less than $\lfloor n/(d+1) \rfloor$ points of S then these $(d+1)r+q$ points can be substituted by q points.*

If this conjecture is proved then we will be able to discard at least a fraction of points in each iteration and a centrepoint in d dimensions can be found in linear time.

Chapter 6

Conclusions

An optimal algorithm has been described in this thesis for computing the intersection radius of a given set of planar objects containing points, straight lines, rays, line segments, wedges, half planes and planes. We have used a new approach of allowing addition of objects in the pruning process while ensuring a simultaneous net reduction of the input set. This provides one more example of the power and versatility of the prune and search paradigm. This algorithm has applications in many practical stabbing radii problems.

We have also presented an optimal algorithm for computing a centrepoint for a planar finite set of points. This settles a long standing problem in computational geometry. It may be worthwhile to point out that an approximate centrepoint suffices in many applications but this exact algorithm, because of its efficiency, can very well replace the existing approximate centrepoint algorithms in two dimensions.

A new technique has been presented to solve geometric optimisation problems using prune and search paradigm. We have been able to do this by the synthesis of parametric searching and prune and search techniques. We have applied this technique to compute ham-sandwich cuts and centrepoints in higher dimensions. This technique has also been applied to the well known problem of linear programming. The results obtained match the best results extant for these problems.

6.1 Further Research Problems

It would be worth investigating if there exist similar algorithms as presented in this thesis to compute intersection radius of objects in higher dimensions. The weighted version of this problem also remains unsolved. The intersection radius for a set of simple polygons cannot be computed using the same algorithm because of the inherent non-convexity of the distance function.

Just as there are centrepoints, we can have other centre objects like centreline, centrecircle etc. The computation of these remain an open problem. The weighted version of the centrepoint problem also remains open. The other problems for a finite set of simple polygons in the plane to look at are area-centrepoints, perimeter-centrepoints and the like. The weighted centrepoint can be addressed as a special case of these problems.

In this thesis, a centrepoint of an explicit set of points is computed in linear time. However, the centrepoint can not be computed within the same time bounds if the point set is implicit, such as the set of intersection points of straight line defined by any two points of a set of points in $\Re^3$ with a fixed plane.

In the technique of parametric prune and search the open area of research is to provide efficient algorithms for pruning. The pruning technique described in this thesis has a doubly exponent factor of d in its time complexity which renders the algorithm impractical for higher dimensions. AKS sorting networks have also large constants of proportionality in their complexities. In practice, to solve the problem discussed in this thesis faster, randomised algorithms can be used. The other important question is how this technique can be applied to other parametric searching algorithms as well. It seems that the method presented in the thesis is a particular case of Mataousek's result[19]. It remains to be seen if this technique can be applied to non-linear programming problems as well.

Furthermore, AKS sorting networks have built-in comparators that take two inputs each. The generalisation of comparison for points in higher dimensional space is their rotational order. So, if a network is designed which has such rotational ordering elements in place of comparators then it can be applied to many more problems using a similar technique.

Bibliography

[1] M. Ajtai, J. Komlos, and E. Szemeredi. An $O(n \log n)$ sorting network. *Combinatorica*, 3:1–19, 1983.

[2] B. K. Bhattacharya, J. Czyzowicz, P. Egyed, G. Toussaint, I. Stojmenovic, and J. Urrutia. Computing shortest transversals of set. In *Proc. of the Seventh Annual ACM Symp. on Computational Geometry*, pages 71–80, 1991.

[3] B. K. Bhattacharya, S. Jadhav, A. Mukhopadhyay, and J. M. Robert. Optimal algorithms for some smallest intersection radius problems. In *Proc. of the Seventh Annual ACM Symp. on Computational Geometry*, pages 81–88, 1991.

[4] B. K. Bhattacharya and G. T. Toussaint. Computing shortest transversals. Technical Report SOCS 90.6, McGill University, April 1990.

[5] K. Q. Brown. Fast intersection of half spaces. Technical Report CMU-CS-78-129, Carnegie Mellon University, Pittsberg, 1978.

[6] G. Chrystal. On the problem to construct the minimum circle enclosing n given points in the plane. In *Proc. Edinberg Math. Soc.*, volume 3, pages 30–33, 1885.

[7] K. L. Clarkson. Linear programming in $O(3^{(k+1)^2})$ time. *Information Processing Letters*, 22:21–24, 1986.

[8] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.

[9] R. Cole, M. Sharir, and C. Yap. On k-hulls and related problems. *SIAM J. Comput.*, 16:61–77, 1987.

[10] D. L. Donoho and M. Gasko. Breakdown properties of location estimates based on halfspace depth and projected outlyingness. *The Annals of Statistics*, 20(4):1803–1827, 1992.

[11] M. E. Dyer. Linear-time algorithm for two- and three-variable linear programs. *SIAM J. Computing*, 13:31–45, 1984.

[12] M. E. Dyer. On a multidimensional search technique and its application to the Euclidean 1-center problem. *SIAM J. Computing*, 15:725–738, 1986.

[13] H. Edelsbrunner. *Algorithms in Computational Geometry.* Springer Verlag, 1987.

[14] M. T. Goodrich and J. S. Snoeyink. Stabbing parallel segments with a convex polygon. In F. Dehne, J.R.Sack, and N. Santaroo, editors, *Procs. of the Workshop on Algorithms and Data Structures*, pages 231–242. Lecture notes in Computer Science 382, Springer Verlag, 1989.

[15] M. E. Houle, H. Imai, K. Imai, and J. M. Robert. Weighted orthogonal linear L_∞-approximation and applications. In F. Dehne, J.R.Sack, and N. Santaroo, editors, *Procs. of the Workshop on Algorithms and Data Structures*, pages 183–191. Lecture notes in Computer Science 382, Springer Verlag, 1989.

[16] S. Jadhav and A. Mukhopadhyay. Designing optimal geometric algorithms using partial sorting networks. In *Proc. of the Third National Seminar on Theoretical Computer Science, India*, 1993.

[17] Chi-Yuan Lo, Jiří Matoušek, and William Steiger. Ham-Sandwich Cuts in R^d. In *Proc. 24th Annual STOC Conference*, pages 539–545, 1992.

[18] Chi-Yuan Lo and William Steiger. An optimal-time algorithm for ham-sandwich cuts in the plane. In *Proc. of the Second Canadian Conference on Computational Geometry*, pages 5–9, 1991.

[19] Jiří Matoušek. Approximations and optimal geometric divide-and-conquer. In *Proc. 23rd Ann. ACM Symposium on Theory of Computing*, pages 505–511, 1991.

[20] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):751–758, 1983.

[21] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Computing*, 12:759–776, 1983.

[22] N. Megiddo. Linear programming in linear time when the dimension is fixed. *JACM*, 31(1):114–127, 1984.

[23] N. Megiddo. Partitioning with two lines in the plane. *J. Algorithms*, 3:430–433, 1985.

[24] H. Meijer and D. Rappaport. Minimum polygon covers of parallel line segments. Technical Report CISC 90-279, Queen's University, Canada, 1990.

[25] G. L. Miller, S.-H. Teng, and S. A. Vavasis. Density graphs and seperators. In *2nd ACM-SIAM Symp. Discrete Algorithms*, pages 538–547, 1991.

[26] G. L. Miller and W. Thurston. Seperators in two and three dimensions. In *22nd ACM Symp. theory of Computing*, pages 300–309, 1990.

[27] A. Mukhopadhyay, C. Kumar, and B. Bhattacharya. Computing an area-optimal convex polygonal stabber of a set of parallel line segments. In *Proc. of 5th Canadian Conference on Computational Geometry, Waterloo, Canada*, 1993.

[28] R. Seidel. Linear programming and convex hull made easy. In *Proc. of the Sixth Annual ACM Symp. on Computational Geometry*, pages 211–215, 1990.

[29] M. I. Shamos. *Computational Geometry.* PhD thesis, Department of Computer Science, 1978.

[30] J. J. Sylvester. A question in the geometry situation. *Quart. J. Pure Appl. Math.*, 1:79, 1857.

[31] J. J. Sylvester. On Poncelet's approximate linear valuation of the surd forms. *Philosophical Magazine*, 20:203–222, 1860.

[32] S.-H. Teng. *Points, Spheres and Seperators: A unified geometric approach to graph partitioning.* PhD thesis, Carnegie-Mellon University, School of Computer Science, 1991. Tech. Rep. CMU-CS-91-184.

[33] H. Tverberg. A generalization of Radon's theorem. *Journal London Math. Soc.*, 41:123–128, 1966.

[34] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science, Proceedings,1991, VIII*, pages 359–370. Lecture Notes in Computer Science 555, Springer Verlag, 1991.

[35] I. M. Yaglom and V. G. Boltyanskii. *Convex Figures.* Holt, Rinehart and Winston, 1961.

[36] F. F. Yao. A 3-space partition and its application. In *15th ACM Symp. Theory of Computing*, pages 258–263, 1983.